IMPLEMENTADI *DELAY TOLERANT NETWORK (DTN)* DENGAN MENGGUNAKAN *ALGORITME HIERARCHICAL TOKEN BUCKET* (HTB) UNTUK SELEKSI *NODE ROUTING MULTI COPY*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Hidayatus Syafa'ah
NIM:135150201111229

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

Implementasi Delay Tolerant Network (DTN) Dengan Menggunakan Algoritme Hierarcichal Token Bucket (HTB) Untuk Seleksi *Node Routing Multi copy*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Hidayatus Syafa'ah
NIM: 135150201111229

Skripsi ini telah diuji dan dinyatakan lulus pada
30 Juli 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Rakhmadhany Primananda, S.T. M.Kom	**Reza Andria Siregar, S.T., M.Kom**
NIK: 201609 86040 6 1001	**NIP: 19790621 200604 1 003**

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 197105182003121001

ii

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, didalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik disuatu perguruan tinggi dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 30 Juli 2018

Hidayatus Syafa'ah

NIM: 135150201111229

KATA PENGANTAR

Dengan menyebut nama Allah SWT yang maha pengasih dan maha penyayang. Penulis panjatkan Puji syukur atas kehadirat Allah SWT karena dengan rahmat dan hidayah-Nya, penulis dapat menyelesaikan skripsi yang berjudul **"IMPLEMENTASI *DELAY TOLERANT NETWORK* (DTN) DENGAN MENGGUNAKAN *ALGORITME HIERARCHICAL TOKEN BUCKET* (HTB) UNTUK SELEKSI *NODE ROUTING MULTI COPY*".** Skripsi ini disusun untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Komputer di Fakultas Ilmu Komputer Universitas Brawijaya Malang.

Penulis menyadari bahwa, skripsi ini tidak akan berhasil tanpa bantuan dari beberapa pihak yang memberi semangat, kritik, saran, bimbingan dan doa. Oleh karena itu ucapan terima kasih penulis sampaikan kepada:

1. Allah S.W.T atas berkah, rahmat dan hidayahnya yang senantiasa menyertai dalam pelaksanaan skripsi ini.
2. Bapak Rakhmadhany Primananda, S.T. M.Kom selaku dosen pembibing I yang telah sabar membimbing dan mengarahkan penulis sehingga dapat menyelesaikan skripsi ini.
3. Bapak Reza Andria Siregar, S.T., M.Kom selaku dosen pembimbing II yang telah bersabar dengan membimbing dan mengarahkan penulis sehingga dapat menyelesaikan skripsi ini.
4. Ayah Yusuf, S.Pd dan Ibu Supartik atas segala nasehat, kasih sayang, perhatian dan kesabarannya dalam mendidik penulis, serta yang senantiasa tiada henti-hentinya memberikan doa dan semangat demi terselesaikannya skripsi ini.
5. Anak-anak kos "Veteran Dalam 5D" dan teman-teman seperjuangan di Fakultas Ilmu Komputer "Liza, Octa, Ervan, Arif, Guntur dan Caca" yang senantiasa memberikan support dan masukan dalam perkuliahan.
6. Seluruh teman – teman Informatika angkatan 2013 Universitas Brawijaya yang telah banyak memberi masukan dan semangat kepada penulis sehingga terselesaikan skripsi ini.

Semoga jasa dan amal baik mendapatkan balasan dari Allah SWT. Akhir kata penulis berharap skripsi ini dapat membawa manfaat bagi semua pihak yang menggunakannya terutama mahasiswa Fakultas Ilmu Komputer Universitas Brawijaya.

Malang, 30 Juli 2018

Penulis
syafaifa29@gmail.com

ABSTRAK

Delay Tolerant Network adalah jaringan yang digunakan untuk komunikasi jarak jauh, dengan *layer* tambahan yaitu *bundle layer*. Pada *Delay Tolerant Network, routing* menjadi tantangan dalam penerapannya, hal ini disebabkan penggunakan *bandwidth* dan *buffer* yang terbatas, sehingga diperlukan manajemen *bandwidth* untuk mengatur kelebihan beban pada *buffer* menggunakan *Algoritme Hierarchical Token Bucket. Routing* digunakan untuk merutekan pesan dari *node* sumber ke *node* tujuan. Terdapat dua kategori *protocol routing* berdasarkan jumlah salinan pesan yaitu *singlecopy routing* dan *multi copy routing. Multi copy routing* yaitu pesan akan diteruskan pada setiap *node* di jalur yang sudah ditentukan. *Routing* ini mampu meningkatkan kinerja jaringan dalam memaksimalkan rasio pengiriman pesan dan minimum *delay. Algoritme Hierarchical Token Bucket* digunakan untuk mengontrol penggunaan *bandwidth* terhadap *link* yang diberikan kepada *client* dan membatasi *download* dan *upload client.* Pada penelitian ini, *Algoritme Hierarchical Token Bucket* diimplementasikan menggunakan simulasi *ONE Simulator* dan bahasa pemrograman *JAVA* pada *Eclipse*. Hasil pengujian dianalisis menggunakan parameter *average latency, overhead ratio, delivery probability* dan *average hop count* dengan skenario jumlah *node* 50, 100, 150 dan 200, kecepatan *node* 20-160km/jam dan ukuran pesan 1MB. Dari hasil pengujian diperoleh bahwa, implementasi *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy* bekerja dengan baik, karena menerapkan penjadwalan pengiriman pesan berdasarkan parameter *ceil* dan *rate.* Hal ini bisa dilihat dari hasil parameter uji *overhead ratio* sebesar 15.9737ms, nilai *delivery probabilty* sebesar 0.9504% dan nilai *average hop count* sebesar 20.5900ms dan *average latency* sebesar 220.2333ms.

Kata kunci: *Delay Tolerant Network, routing protocol, MaxProp, manajemen bandwidth, Algoritme Hierarchical Token Bucket, Bundle Layer.*

ABSTRACT

Delay Tolerant Network is network used for long distatnce communication with layer addition, whic is bundle layer. In Delay Tolerant Netwrok, routing becomes a challenge in its implementation. It is caused by the use of limited bandwidth and buffer thus it need bandwidth management in order to arranged overload in buffer using Hierarchical Token Bucket Algorithm. Routing is used to route messages from the source node to the destination node. There are two categories of routing protocols based on the number of message copies, namely singlecopy routing and multi routing copy. Multi copy routing is, the message will be forwarded to each node on the specified path. This routing is able to improve network performance in maximizing the delivery message and minimum delay. Hierarchical Token Bucket Algorithm used to control the use of bandwidth towards link given to client and limit the download and upload conducted by client. In this research, Hierarchical Token Bucket Algorithm implemented by using ONE Simulator simulation and JAVA programming languange in Eclipse. The result of test then analysed by using average latency, overhead ratio, delivery probabilty and average hop count parameter with scenario number of node 50, 100, 150 and 200 node and node speed 20-160km/hourand message size of 1MB. From the result of test, it is obtained that the implementation of Hierarchical Token Bucket Algorithm for node routing multi copy selection runs appropriaately due to it implements message delivery schedule based on ceil and rate parameter. It can be seen from the result of parameter test of overhead ratio for 15.9737ms, score of delivery probabilty for 0.9504%, score average hop count score is 20.5900ms and average latency score is 220.2333ms.

Keyword: *Delay Tolerant Network, routing protocol, MaxProp, management bandwidth, Algoritme Hierarchical Token Bucket, Bundle Layer.*

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Delay Tolerant Network adalah jaringan yang digunakan untuk komunikasi jarak jauh. Pada *Delay Tolerant Network, routing* menjadi tantangan dalam penerapanya, hal ini disebabkan penggunaan *bandwith dan buffer* yang terbatas (Abdillah, et al., 2015). Routing digunakan untuk merutekan pesan dari *node* sumber ke *node* tujuan dengan bergantung pada skenario *routing* yang digunakan. Terdapat dua mekanisme *routing* dalam *Delay Tolerant Network* seperti *protocol First Contact dan Direct Delivery* yang menerapkan *SingleCopy Strategy* (Wang, et al., 2010). Mekanisme *routing* yang lain adalah *multi copy strategy* yaitu pesan diteruskan pada setiap *node* pada jalur yang ada. Strategi ini mampu meningkatkan kinerja jaringan dalam memaksimalkan rasio pengiriman dan meminimalkan *delay*, namun dibutuhkan sumber daya tambahan seperti *bandwidth, energy* dan *buffer. Routing multi copy* lebih banyak diterapkan karena kinerja *routing* yang lebih handal (Muis, et al., 2018). Namun pada penelitian yang akan dilakukan akan menerapkan *routing multi copy* yaitu *protocol MaxProp*

Cara kerja *protocol MaxProp* yaitu pesan akan dikirimkan pada *node* dengan nilai *path cost* tertinggi dan memberikan prioritas pada pesan yang baru datang. Semakin banyak *node* sumber bertemu dengan *node* perantara, maka *cost* pengiriman dari *node* sumber pada *node* tujuan akan bertambah. Pada penelitian ini juga akan dilakukan seleksi *node* untuk menentukan pesan akan dikirimkan pada *node* tujuan yang sudah ditentukan berdasarkan mekanisme *two hop routing*. *Two hop routing* adalah pesan hanya akan dikirimkan pada *node* yang menjadi tujuan dari *node* sumber (Jones & Ward, 2006).

Hierarchical Token Bucket adalah sistem yang digunakan untuk manajemen *bandwidth*. Dimana jumlah *bandwidth* yang diminta tidak boleh melebihi dari kapasitas *bandwidth* yang disediakan oleh *sistem*. *Bandwidth* dan *buffer* pada *Delay Tolerant Network* sangat terbatas, sehingga diperlukan manajemen *bandwidth* untuk mengatur kelebihan beban pada *buffer* menggunakan *Algoritme Hierarchical Token Bucket. Algoritme* ini dipilih karena, mampu membuat *node* mendapatkan *bandwidth* diantara nilai *rate* dan nilai *ceil. Algoritme Hierarchical Token Bucket* berperan dalam mengontrol penggunaan *bandwidth* terhadap *link* yang diberikan kepada *client* dan membatasi *download* dan *upload clien* (Wijaya & Handoko, 2013).

JLS (Jalur Lintas Selatan) Malang merupakan jalur yang menghubungkan pantai Balekambang dan Pantai Sendangbiru. Jalur sepanjang 25 kilometer (km) ini menembus area konservasi lahan pertanian dengan memotong bukit dan melintasi pegunungan kapur selatan. Jumlah pengunjung semakin meningkat dengan adanya jalur JLS ini. Namun, kondisi jaringan pada jalur JLS ini sama sekali tidak baik, mengingat jalur ini yang dulunya area konservatif. Hal ini disebabkan tidak adanya pemancar sinyal dan jaringan yang menyebabkan pengguna internet tidak bisa mengaksesnya di Jalur Lintas Selatan ini. Hal ini lah yang mendukung untuk dilakukannya simulasi pada Jalur Lintas Selatan dengan

menerapkan *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy* (Nugraha, 2016).

Penelitian sebelumnya dengan judul *Flow Based Transmission Scheduling in Constrained Delay Tolerant Network* pada tahun 2012. Penelitian tersebut membandingkan *algoritma routing Flow Based Transmission Scheduling, MED, MaxProp dan RAPID pada Delay Tolerant Network*. Dimana parameter yang di uji *number of session, number of student, lenght of time slot, time to live of packet, simulation time, packet production rates dan bandwidth.* Pada penelitian ini lebih difokuskan pada penjadwalan *bandwidth* dengan menggunakan nilai minimum yang diterapkan pada DTN. FBTS digunakan untuk menghindari kemacetan dan estimasi waktu pengiriman. Pada penelitian ini menggunakan *protocol routing MED, MaxProp dan RAPID* dengan *efisiensi delivery ratio, delay dan hop pada FBTS*. Pada penelitian ini implementasi dilakukan dengan mengunakan bahasa pemrograman java pada *Eclipse.* Pada penelitian ini terbukti bahwa Rasio pengiriman pada *FBTS* 24% lebih besar dari protokol *RAPID* dan 55% lebih besar dari protoko *MaxProp*. Untuk *delay* pengiriman, pada protokol *MED, RAPID* dan *MaxProp* 56% lebih panjang dari *FBTS* (Yang, et al., 2012).

Dari permasalahan yang telah dijelaskan dan dari penelitian yang sudah ada, penulis berniat mengembangkan penelitian *Flow Based Transmission Scheduling in Constrained Delay Tolerant Network*. Penelitian yang akan di lakukan berfokus pada seleksi *node* dengan menggunakan *Algoritme Hierarchical Token Bucket* dan menggunakan *routing multi copy*. *Algoritme* ini dipilih karena, mampu membuat *node* mendapatkan *bandwidth* diantara nilai *rate* dan nilai *ceil* dan setiap pesan yang terkirim akan melewati antrian tanpa adanya *delay*. Dengan skenario jumlah *node* sebesar 50, 100, 150 *node* dan 200 *node*. Ukuran pesan yang digunakan sebesar 1MB dan kecepatan *node* (20-40), (40-80), (80-160) km/jam. Seleksi *node* digunakan untuk menentukan *node* mana yang akan dikirimi pesan oleh *node* sumber pada *node* tujuan. Pada penelitian ini, akan menggunakan simulasi *ONE Simulator* dan bahasa pemrograman *JAVA* pada *Eclipse* untuk mengimplementasikan *Algoritme Hierarchical Token Bucket.*

1.2 Rumusan Masalah

Dari latar belakang diatas, dapat dirumuskan beberapa permasalahan dalam penelitian ini, diantaranya sebagai berikut:

1. Bagaimanakah implementasi *Delay Tolerant Network* menggunakan *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy*?
2. Bagaimanakah implementasi *Delay Tolerant Network* tanpa menggunakan *Algoritme Hierarchical Token Bucket?*
3. Bagaimanakah proses *manajement bandwidth* dilakukan dengan skenario jumlah *node* 50, 100, 150 dan 200 *node?*
4. Bagaimanakah hasil seleksi *node* terhadap parameter uji pada *Algoritme Hierarchical Token Bucket?*

1.3 Tujuan

Penelitian yang akan dilakukan memiliki beberapa tujuan, yaitu:

1. Merancang dan mensimulasikan *Delay Tolerant Network* dengan menggunakan *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy* pada Jalur Lintas Selatan (JLS) Malang menggunakan *Opportunistic Network Environtment (ONE) Simulator*.
2. Menganalisis hasil seleksi *node* untuk *routing multi copy* dengan menggunakan *Algoritme Hierarchical Token Bucket* berdasarkan parameter performansi terhadap parameter uji yang sudah ditentukan.
3. Menganalisis hasil *protocol MaxProp* tanpa menggunakan *Algoritme Hierarchical Token Bucket* berdasarkan parameter performansi terhadap parameter uji yang sudah ditentukan.
4. Menganalisis hasil manajemen *bandwidth* berdasarkan jumlah *node* sebanyak 50, 100, 150 dan 200 *node.*

1.4 Manfaat

Dalam penelitian ini memiliki manfaat yang diharapkan untuk penulis dan pengembang penelitian yang lain, yaitu:

1. Dapat dijadikan sebuah referensi sebagai bahan penelitian selanjutnya yang berkaitan dengan seleksi *node* pada *Algoritme Hierarchical Token Bucket* pada *routing multi copy*.
2. Dapat digunakan seorang *administrator* jaringan untuk dijadikan alternatif model jaringan dan referensi dalam seleksi *node*.

1.5 Batasan Masalah

Masalah dalam penelitian ini akan dirumuskan kedalam beberapa batasan, yang ditujukan supaya tidak menyimpang dari apa yang peneliti lakukan dan lebih terfokus. Berikut batasan masalah pada penelitian ini, yaitu sebagai berikut:

1. *Protocol routing* yang digunakan yaitu *protocol maxprop.*
2. Simulasi jaringan menggunakan *ONE Simulator*1.4.1 RC2.
3. Pengolahan peta menggunakan perangkat lunak *OpenJUMP* 1.8.0.
4. Pengunjung akan dijadikan *node* dalam penelitian ini.
5. Simulasi hanya akan dilakukan untuk Jalur Lintas Selatan (JLS) Malang.
6. Metrik pengujian yang digunakan seperti *overhead ratio, average latency, delivery probability dan average hop count.*
7. Parameter yang diuji adalah Jumlah *node* 50, 100, 150 dan 200 *node*, *routing protocol* yaitu *protocol maxprop*, ukuran pesan yang digunakan sebesar 1MB, waktu simulasi yang digunakan yaitu 3600s dan kecepatan *node* yang digunakan yaitu 20-40km/jam, 40-80km/jam dan 80-160km/jam.
8. *Implementasi Delay Tolerant Network* dengan menerapkan *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy* dan *protocol Maxprop*.

9. Ukuran *buffer* yang digunakan yaitu 1GB dan maksimal *bandwidth* yang bisa diminta 6Mbps.
10. Manajemen *bandwidth* menggunakan *Algoritme Hierarchical Token Bucket* dengan proses *duplicate* untuk 50, 100, 150 dan 200 *node.*

1.6 Sistematika Pembahasan

Penulisan struktur skripsi atau sistematika pembahasan yang telah penulis susun dapat menggunakan kerangka sebagai berikut:

BAB I PENDAHULUAN

Pada bab satu memuat tentang latar belakang, rumusan masalah, tujuan, manfaat, batasan masalah dan sistematika pembahasan.

BAB II LANDASAN KEPUSTAKAAAN

Pada bab dua menguraikan kajian pustaka serta dasar teori yang diigunakan sebagai dasar dari penelitian ini.

BAB III METODOLOGI

Pada bab tiga ini membahas tentang metodologi yang digunakan dalam penelitian, bagaimana tahap-tahap dari penelitian ini mulai dari studi literatur, analisis kebutuhan, perancangan, implementasi, pengujian dan analisis hasil dan kesimpulan.

BAB IV REKAYASA KEBUTUHAN

Pada bab empat ini membahas tentang deskripsi umum sistem dan kebutuhan yang diperlukan untuk pengujian. Pada tahap ini terdapat dua kebutuhan yaitu kebutuhan perangkat keras dan kebutuhan perangkat lunak.

BAB V PERANCANGAN DAN IMPLEMENTASI

Pada bab lima ini menjelaskan proses perancangan dan *implementasi* untuk *protocol MaxProp* dan *implementasi algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy*. Terdapat tiga tahap untuk proses perancangan dan implementasi yaitu perancangan protokol *maxprop,* perancangan *Algoritme HTB* dan perancangan *sistem.*

BAB VI PENGUJIAN DAN HASIL

Pada bab enam akan dilakukan pengujian dan hasil untuk mengetahui kinerja dari *protocol* dan *Algoritme* yang sudah diterapkan pada tahap *implementasi*.

BAB VII KESIMPULAN

Pada bab tuju ini memuat kesimpulan dari penelitian yang sudah dilakukan serta saran yang dapat bermanfaat untuk penelitian yang akan dilakukan selanjutnya.

BAB 2 LANDASAN KEPUSTAKAAN

Pada bab 2 berisikan landasan kepustakaan yang akan membahas mengenai penelitian sebelumnya dan permasalahan yang akan diselesaikan dan mengenai teori yang berkaitan dengan topik yang meliputi landasan pustaka dan dasar teori.

2.1 Tinjauan Pustaka

Tinjauan pustaka pada penelitian ini, akan membahas penelitian sebelumnya dengan penelitian yang akan diusulkan. Penelitian sebelumnya masih dengan objek yang sama yaitu mengenai *Delay Tolerant Network* tapi dengan fokus penelitian yang berbeda. Perbandingan penelitian dilakukan dengan membandingkan parameter dan *protocol routing* yang digunakan serta *output* yang dihasilkan. Tabel 2.1 menjelaskan perbandingan penelitian yang akan dilakukan, yaitu:

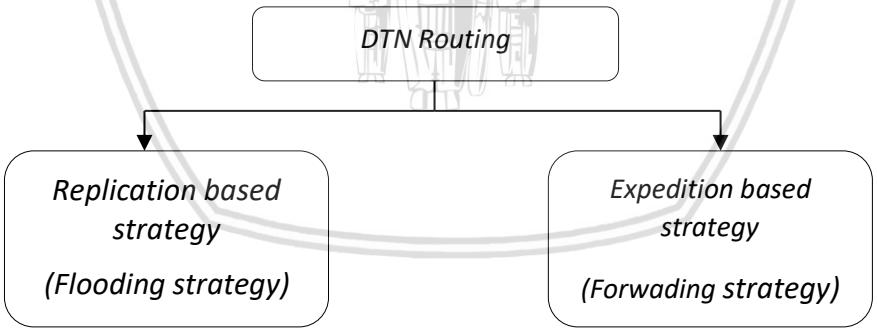
Tabel 2. 1 Tabel Perbandingan Penelitian Penulisan dengan Penelitian Terkait

No	Nama peneliti, Tahun dan Judul	Persamaan	Perbedaan	
			Penelitian Terdahulu	Rencana Penelitian
1.	*Yang, et al.,[2012] Flow Based Transmission Scheduling in Constrained Delay Tolerant Network*	Melakukan analisis pada *protocol routing MED, MaxProp an RAPID* pada *Delay Tolerant Network* dengan menggunakan *algoritme Flow Based Transmission Scheduling*	implementasi *algoritme Flow Based Transmission Scheduling* pada *Delay Tolerant Network* dengan menggunakan bahasa pemrograman JAVA pada *Eclipse*	Menerapkan *algoritme Hierarchical Token Bucket* untuk seleksi *node routing multy copy* yaitu *protocol Maxprop.*
2.	*Krifa & Barakat[2008] An Optimal Joint Scheduling and Drop Policy for Delay Tolerant Networks*	Melakukan analisis pada *protocol routing* yaitu *GBSD, HBSD, RAPID dan Epidemic* yang berfokus pada *scheduling* dan *manajemen buffer* pada *Delay Tolerant*	Melakukan analisis pada *protocol r outing GBSD, HBSD, MaxProp dan Epidemic* dengan menggunakan beberapa metrik	Menerapkan *algoritme Hierarchical Token Bucket* untuk seleksi node *routing multy copy* yaitu *protocol Maxprop dengan* menggunakan

		Network. *Protocol routing* GBSD dan HBSD lebih unggul ketika *unlimited buffer dan limited bandwidth.*	performansi yaitu waktu simulasi, area simulasi, number of , *average speed, TTL dan CBR Interval.*	metrik performansi yaitu Jumlah *node*, *routing protocol,* ukuran pesan, waktu simulasi dan kecepatan *node* .
3.	Kimura &Premachandra [2016]	Melakukan seleksi *node* pada *routing multi copy*	Melakukan seleksi *node* pada *routing multi copy* yaitu *protocol Spray and Wait*	Melakukan seleksi *node* pada *routing multi copy* dengan menggunakan *Algoritme Hierarchical Token Bucket*.

2.2 *Delay Tolerant Network* (DTN)

Tujuan *Delay Tolerant Network* adalah merutekan pengiriman pesan pada jaringan *mobile* dan jaringan luar angkasa (Patel & Gondaliya, 2015). Gambar 2.1 adalah kategori *routing* pada *Delay Tolerant Network ,* yaitu:



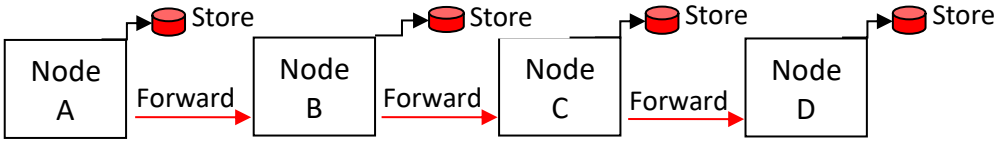
Gambar 2. 1 *Delay Tolerant Network Routing Strategy*

Sumber : [(Alaoui, et al., 2015)]

Strategi *routing* tergantung pada *node* dalam mentransmisikan data seperti: *flooding strategy* yaitu strategi dengan membuat banyak salinan pesan untuk diteruskan pada tujuan dan *forwarding strategy* yaitu penggunaan mekanisme yang berbeda untuk pemilihan relay *node* dengan penyimpanan yang terbatas (Alaoui, et al., 2015).

2.2.1 Cara Kerja *Delay Tolerant Network*

Pada *Delay Tolerant Network* (DTN) menerapkan metode *Store, Carry and Forward* (SCF). Dengan menggunakan metode tersebut, *node* sumber mampu berkomunikasi dengan *node* tujuan tanpa saling bertemu. (Muis, et al., 2018).



Gambar 2. 2 *Store Carry Forward*

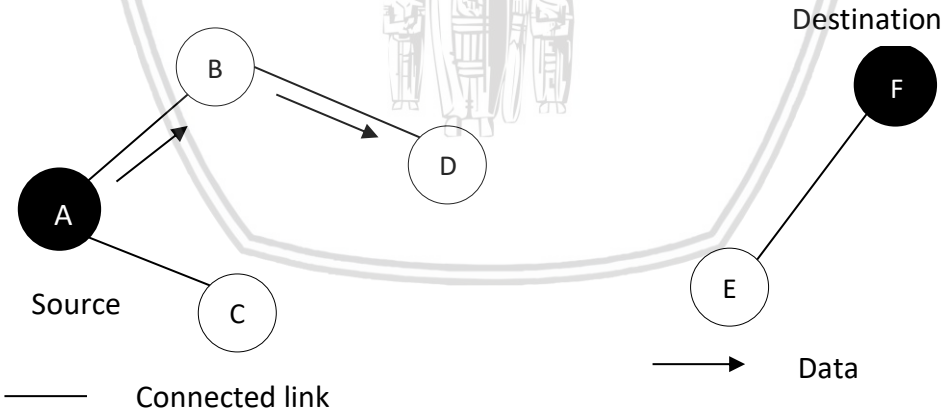
Sumber : [(Muis, et al., 2018)].

1. *Store* adalah proses dimana setiap *node* menyimpan pesan yang masuk pada *Delay Tolerant Network.*
2. *Carry* adalah proses dimana *node* membawa pesan dari sumber ke tujuan.
3. *Forward* adalah proses meneruskan pesan ketika *node* bertemu dengan *node* yang lain setiap kali pengiriman pesan dimulai.

Pada *Delay Tolerant Network* terdapat beberapa mekanisme penggiriman pesan yang diterapkan seperti:

2.2.1.1 *One Hop Routing Example*

Yaitu pesan akan dikirimkan ketika *node* sumber bertemu dengan *node* perantara dan diteruskan pada *node* tujuan. Ketika *node* sumber tidak bertemu dengan *node* perantara, maka pesan akan disimpan sampai bertemu dengan *node* perantara.



Gambar 2. 3 Ilustrasi Cara Kerja DTN dengan *One Hop Routing*

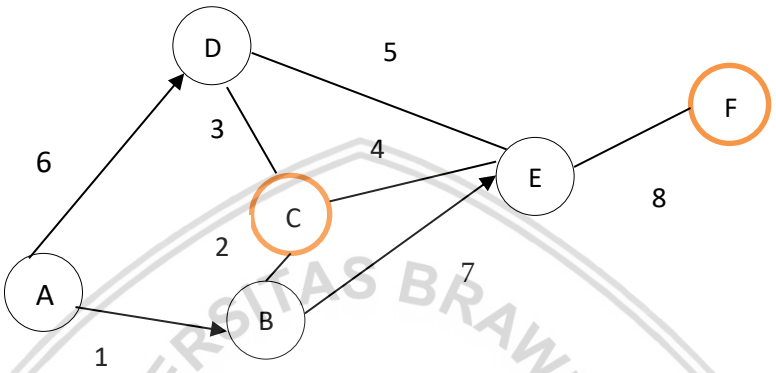
Sumber (Akbi & Wibisono, 2016)

Node A sebagai *node* sumber mengirimkan pesan pada *node* B sebagai perantara, *node* B mengirimkan pesan pada *node* D sebagai perantara selanjutnya, pesan yang diterima oleh *node* D disimpan terlebih dahulu karena, tidak ada perantara selanjutnya yang terbuhung dengan *node* D. Karena *node* D tidak bertemu dengan *node* perantara selanjutnya, maka *node* D menyimpan dan membawa pesan sampai bertemu dengan *node* selanjutnya. Ketika *node* D bertemu dengan *node* perantara selanjutnya yaitu *node* E, maka *node* D akan

mengirimkan pesan pada *node* E dan *node* E akan meneruskan pesan pada *node* F sebagai *node* tujuan.

2.2.1.2 *Two Hop Routing Example*

Yaitu pesan akan dikirimkan hanya pada *node* yang menjadi tujuan dari *node* sumber. Pada *Two hop routing* banyak mengkonsumsi *bandwidth* karena, salinan pesan yang diteruskan bernilai n+1 untuk setiap *node*. Gambar 2.5 menunjukan ilustrasi dari cara kerja pengiriman pesan pada *two hop routing,* yaitu:

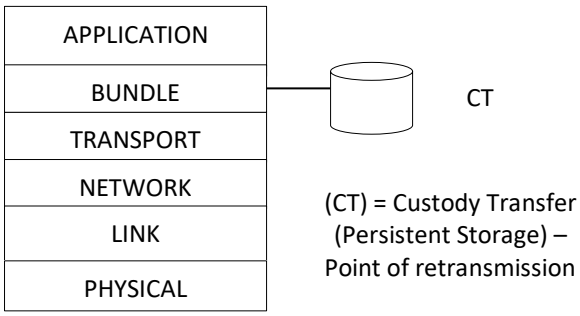


Gambar 2. 4 Mekanisme *Two Hop Routing*
Sumber :[(Jones & Ward, 2006)].

Node A akan mengirim pesan pada *node* E, *node* A akan mengirikan pesan tersebut pada *node* B dan *node* D terlebih dahulu. Ketika *node* B terhubung dengan *node* C, maka pesan tidak akan dikirimkan pada *node* C karena, *node* C bukan tujuan dari *node* A. Namun pesan akan dikirim ketika *node* B terhubung dengan *node* E. *Node* A, B, D dan E akan menerima pesan *node* A bisa mencapai semua melalui *two hop relay* tanpa *node* F.

2.2.2 Arsitektur *Bundle Layer* Pada *Delay Tolerant Network*

Pada *Delay Tolerant Network* terdapat layer tambahan yaitu *Bundle Layer*. *Bundle layer* digunakan untuk menyimpan pesan pada *Delay Tolerant Network* yang dibawa oleh *node* dan diteruskan ketika bertemu dengan *node* yang lain. (Raj & Chezian, 2013). Gambar 2.7 menunjukan arsitektur *bundle layer,* yaitu:



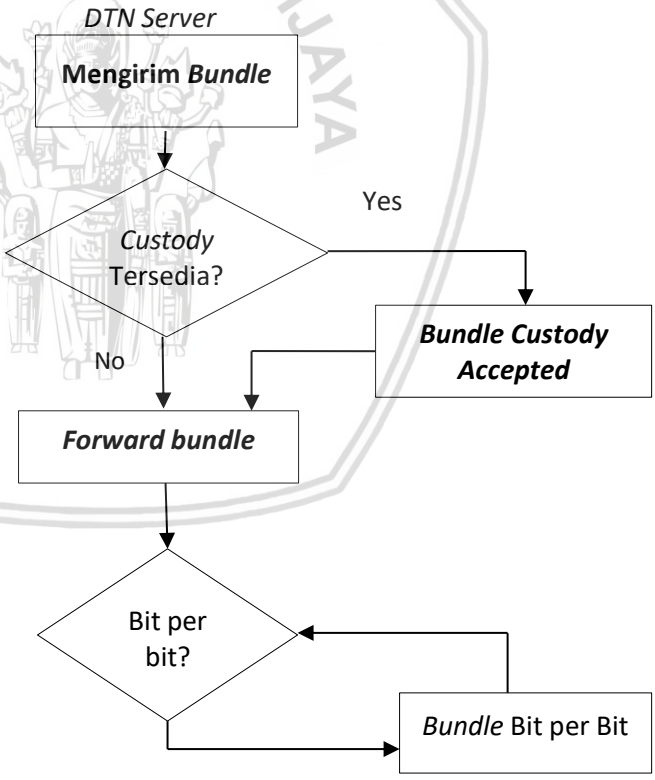
Gambar 2. 5 *Layer* Tambahan Pada *Delay Tolerant Network*
Sumber : [(Raj & Chezian, 2013)]

Kehandalan *bundle layer* dapat dilakukan dengan memilih *protocol transport* melalui dua opsi seperti *end-to-end acknowledgement* dan *custody transfer*. Untuk *node* yang menerima *bundle* akan bertanggung jawab terhadap pengiriman atau disebut *custodians.* Pergerakan *bundle* dari satu *node* ke yang lain disebut *custody transfer.* Ketika jaringan pada pihak *server* terjadi gangguan, maka *bundle* akan disimpan oleh *custodians* pada *database* dan akan dikirim ulang ketika *node* yang berikutnya berada pada jangkauan.

Pada *bundle layer* terdapat 3 proses dalam pengiriman pesan yaitu proses pada *DTN Server, DTN Intermediatte Node* dan *DTN client,* berikut adalah alur kerja dari ketiga proses:

1. *Delay Tolerant Network Server*

Pada tahap ini proses pengiriman dimulai oleh *sserver* dengan meminta koneksi terlebih dahulu sebelum mengirim pesan pada *client.* Yang bertindak sebagai *server* pada *bundle layer* yaitu *sistem* yang menyediakan *bandwidth* untuk *client*. Terdapat status laporan terkirim ketika pesan yang dikirim berhasil diterima oleh *client*. *Bundle* dapat dikirim berupa *bit-per-bit* ketika *socket* transportasi tidak tersedia karena adanya gangguan jaringan. Gambar 2.8 adalah ilustrasi dari *Delay Tolerant Network Serverr,* yaitu:



Gambar 2. 6 Proses Pengiriman *Bundle DTN* Pada *Server*

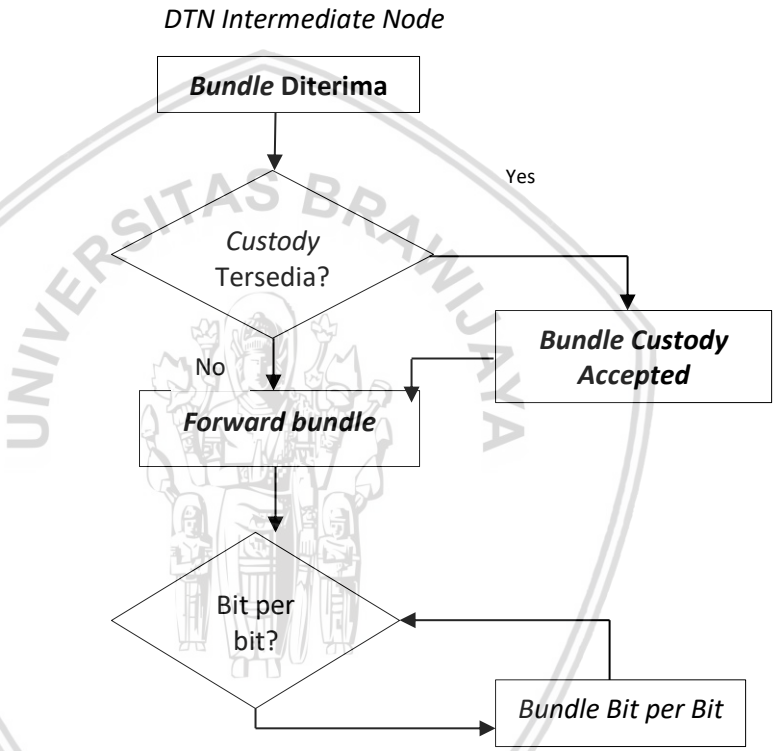
Sumber:[(Caini, et al., 2010)]

Pada tahap *DTN server* proses dimulai dengan mengirim *bundle, bundle* yang dikirim yakni dalam bentuk *bit*. Setelah itu mengecek apakah *custody* tersedia atau tidak. *Custody* adalah *node* yang menerima *bundle.* Hal ini dilakukan untuk memastikan bahwa *server* mengirim *bundle* pada *client* yang

tepat. Ketika koneksi tidak tersedia, *server* akan melakukan *forward bundle* secara langsung pada *client*.

2. Delay Tolerant Network Intermediatte Node

Pada tahap ini dimulai dengan proses menerima *bundle* dari *DTN Server* dengan mengecek *custody* terlebih dahulu untuk memastikan pesan yang diterima dari *server* sudah diterima atau tidak. Pada tahap ini *intermediatte node* berperan sebagai *node* relay untuk mentransmisikan data antara *node* sumber dan *node* tujuan atau yang biasa disebut dengan *gateway* yang berkomunikasi menggunakan jaringan LAN atau WAN. Gambar 2.9 adalah ilustrasi dari *Delay Tolerant Network Intermediatte Node,* yaitu:



Gambar 2. 7 Proses Pengiriman *Bundle DTN* Pada *Intermediatte Node*

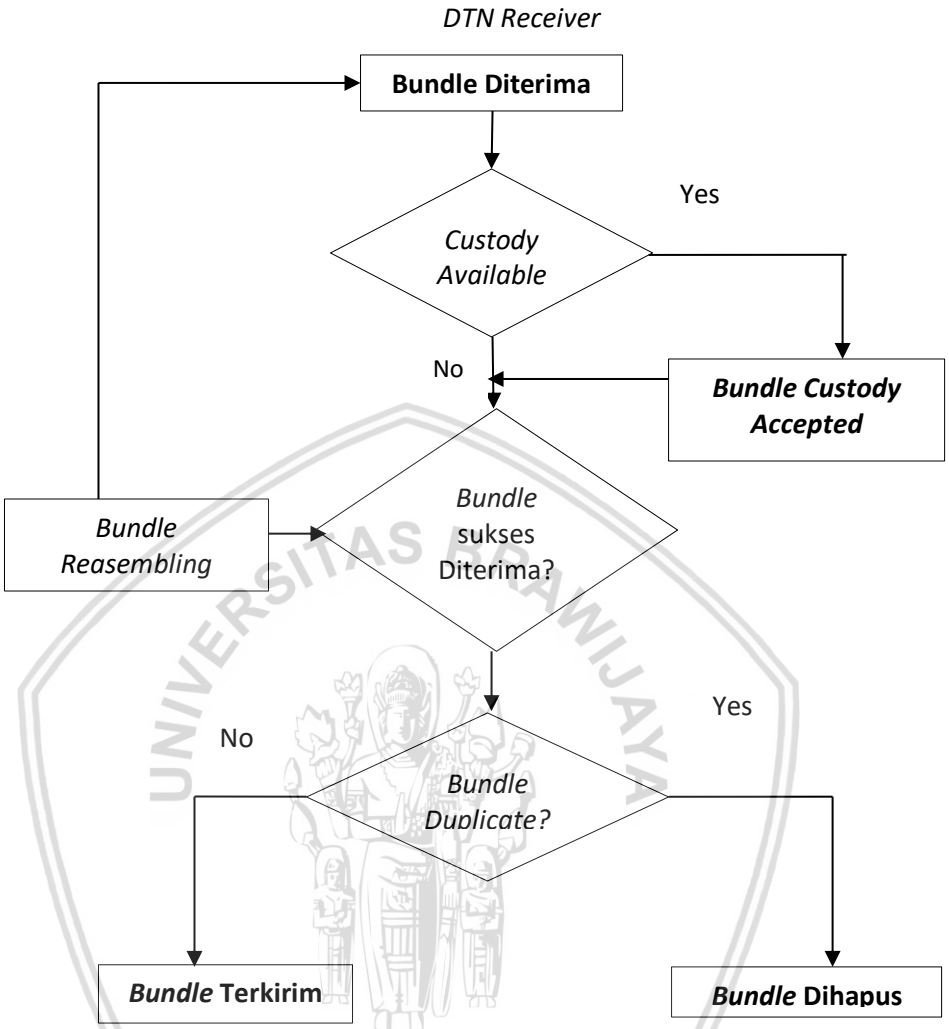
Sumber:[(Caini, et al., 2010)]

Pada tahap *DTN Intermediatte node* proses dimulai dengan menerima *bundle, bundle* yang diterima yakni dalam bentuk bit. Setelah itu mengecek apakah *custody* tersedia atau tidak. Hal ini dilakukan untuk memastikan bahwa *bundle* yang diterima dari *server* yang tepat yang terkoneksi dengan *receiver*. Ketika koneksi tidak tersedia, *server* akan mengirim ulang *bundle.* Ketika koneksi tersedia, *server* akan melakukan *forwading bundle* dalam bentuk *bit per bit.*

3. Delay Tolerant Network Receiver

Pada tahap ini, dimulai dengan proses menerima *bundle* dari *server* dengan melakukan cek koneksi terlebih dahulu. Pihak *Receiver* bisa memilih untuk mengirim *bundle* atau menghapusnya setelah *bundle* di *duplicate*. Yang bertindak sebagai *receiver* pada *bundle layer* yaitu *node* yang akan melakukan

peminjaman *bandwidth.* Gambar 2.10 adalah ilustrasi dari *Delay Tolerant Network Receiver* yaitu:



Gambar 2. 8 Proses Pengiriman *Bundle DTN* Pada *Receiver*

Sumber:[(Caini, et al., 2010)]

Pada tahap *DTN Receiver,* proses dimulai dengan menerima *bundle* dari *DTN server.* Sebelum menerima *bundle,* terlebih dahulu melakukan cek *custody*, hal ini ditujukan agar *bundle* yang diterima sesuai dengan *bundle* yang dikirim oleh *server.* Ketika koneksi sudah tersedia, perlu dilakukan cek *bundle* yang diterima sukses secara keseluruhan atau tidak. *Buffer* yang digunakan dalam penelitian ini sebesar 1GB. Ketika jumlah total *bandwidth* yang diperoleh melebihi dari ukuran *buffer,* maka akan dilakukan proses *duplicate* yaitu proses peminjaman *bandwidth* akan dihentikan karena jumlah *bandwidth* yang digunakan sudah mencapai ukuran *buffer,* yang berarti bahwa *node* yang belum melakukan peminjaman *bandwidth* tidak bisa melakukan peminjaman karena jumlah *bandwidth* yang diperoleh sudah melebihi dari kapasitas *buffer* yang digunakan.

2.2.3 Klasifikasi *Protokol Routing*

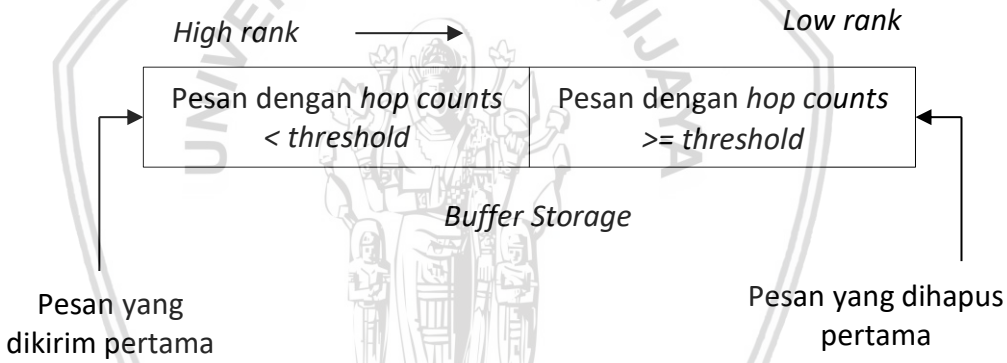
Terdapat dua kategori *protocol routing* berdasarkan jumlah salinan pesan yaitu: *single copy strategy* dan *multi copy strategy*. Pada penelitian yang dilakukan akan menerapkan *routing multi copy* yaitu *protocol MaxProp*.

2.2.3.1 *Multi copy Routing*

Yaitu pesan akan diteruskan pada setiap *node* di jalur yang sudah ditentukan. *Routing* ini mampu meningkatkan kinerja jaringan dalam memaksimalkan rasio pengiriman dan minimum delay, namun dibutuhkan sumber daya tambahan jaringan seperti *bandwidth*, *energy dan buffer* (Muis, et al., 2018).

2.2.3.2 *MaxProp Protocol*

Protocol MaxProp adalah pengiriman pesan pada *protocol* yang dihitung berdasarkan nilai *cost*. *Cost* adalah estimasi pengiriman. *Protocol MaxProp* menggunakan *adaptive threshold* untuk menentukan antrian pesan. *Protocol* ini akan bekerja dengan baik ketika *buffer* yang digunakan besar. Gambar 2.11 adalah strategi dalam pengiriman pesan *protocol MaxProp,* yaitu:



Gambar 2. 9 Strategi *Routing MaxProp*

Sumber: [(Abdillah, 2015)].

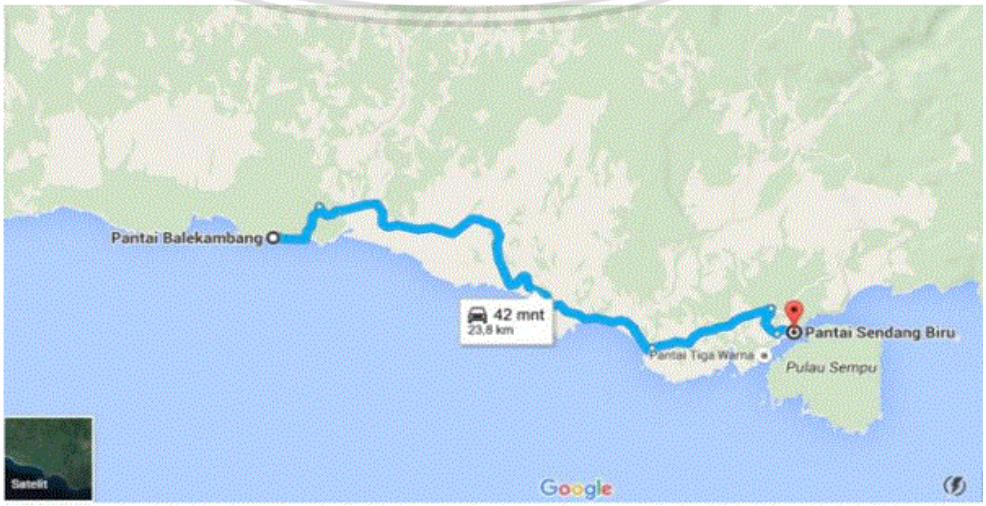
Pada *protocol MaxProp* pesan dengan *hop count* < *threshold* akan dikirimkan terlebih dahulu. *Threshold* bekerja untuk menentukan pesan yang akan dikirim terlebih dahulu. Saat kedua *node* bertemu, pesan dengan nilai *hopcount* < *threshold* akan dikirimkan terlebih dahulu dengan melihat nilai *hopcount* terkecil. Sedangkan pesan dengan nilai *hopcount* ≥ *threshold* akan disimpan. Dengan ukuran *buffer* yang besar kemungkinan pesan dibuang lebih kecil. Ketika pesan dengan nilai *hopcount* < *threshold* sudah dikirimkan semua, maka pesan dengan *hopcount* ≥ *threshold* menjadi prioritas pengiriman dengan melihat nilai *hopcount* yang terkecil. Sehingga kemungkinan pesan terkirim dengan tujuan yang jauh bertambah.

Pesan dengan prioritas tertinggi akan dikirimkan pertama selama waktu pengiriman pesan masih ada. Pesan dengan prioritas rendah akan dihapus dengan tujuan menyediakan tempat untuk pesan yang akan datang.

- *Complementary Mechanisms* adalah penggunaan beberapa mekanisme untuk meningkatkan *delivery rate* dan mengurangi *latency*.
- *Acknowledgment* digunakan untuk mencegah menerima pesan yang sama.
- Pesan yang belum pernah dikirim memiliki prioritas lebih tinggi.
- Pesan yang tidak terkirim akan dikirim berdasarkan prioritasnya.
- Pesan yang sudah terkirim tidak akan dikirim lagi.
- Ketika pesan di*drop* dari *buffer*, maka pesan tidak akan dikirim ulang.
- Salinan pesan sudah pernah dikirim ke *node* tujuan.
- Tidak ada jalur dengan *bandwidth* yang cukup antara *node* sumber dengan *node* tujuan.

2.3 JLS (Jalur Lintas Selatan) Malang

JLS (Jalur Lintas Selatan) yang menghubungkan pantai Balekambang dan Pantai Sendangbiru sudah terhubung. Jalur sepanjang 25 kilometer (km) dengan jarak tempuh 42 menit. Pada jalur lintas selatan malang ini pengunjung untuk setiap minggu nya mencapai 200 orang. Namun semakin banyak nya pengunjung, jaringan *internet* tidak bisa diakses pada jalur ini karena tidak adanya pemancar jaringan. Jalan ini menembus konservasi lahan pertanian dengan memotong bukit dan melintasi pegunungan kapur selatan. JLS memiliki daya tampung dan daya dukung yang harus dipelihara kelestarian lingkungannya karena area ini merupakan area konservatif. Jumlah pengunjung semakin meningkat dengan adanya jalur JLS ini. Namun, kondisi jaringan pada jalur JLS ini sama sekali tidak baik, mengingat jalur ini yang dulunya area konservatif. Hal ini disebabkan karena tidak adanya tower penyedia jaringan yang menyebabkan orang dengan pengguna internet tidak bisa mengaksesnya di jalur JLS tersebut. Permasalahan ini cocok dengan konsep dasar *Delay Toleran Network* dimana, jaringan ini cocok untuk direalisasikan di daerah terpencil karena JLS tersebut menembus jalan konservatif dan lahan pertanian warga sekitar (Nugraha, 2016). Gambar Gambar 2.12 merupakan gambar peta pada JLS (Jalur Lintas Selatan) Malang yaitu:



Gambar 2. 10 Peta Jalur Lintas Selatan
Sumber: [Google Maps)]

2.4 *Hierarchical Token Bucket* (HTB)

Hierarchical Token Bucket adalah disiplin antrian dengan menerapkan *link sharing* secara adil. Konsep *link sharing* adalah ketika suatu *node* meminta *bandwidth* kurang dari jumlah yang ditetapkan, maka sisa *bandwidth* akan didistribusikan pada *node* yang lain. *Algoritme Hierarchical Token Bucket* menggunakan *Token Bucket Filter* (TBF) sebagai estimator, dimana ukuran *bucket* menentukan banyaknya token yang dapat disimpan (Wijaya & Handoko, 2013). *Bucket* adalah tempat untuk menampung pesan, sedangkan token adalah ukuran dari pesan yang dikirimkan.

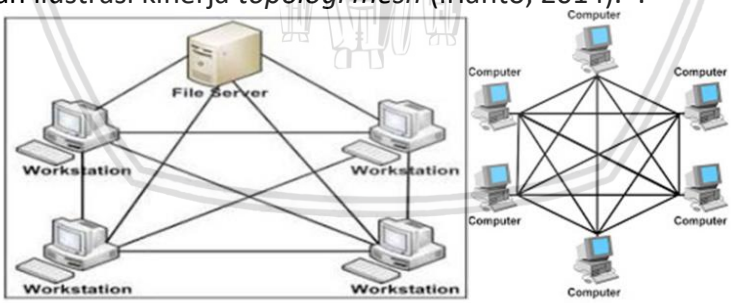
Parameter *ceil* memungkinkan *node* mendapatkan *bandwidth* diantara nilai *rate* dan nilai *ceil*. Parameter ini digunakan sebagai peminjaman *bandwidth* pada node diatasnya selama *bandwidth* yang diperoleh memiliki nilai dibawah *ceil*. Apabila nilai *ceil* sama dengan nilai *rate*, maka *node* tidak diijinkan untuk meminjam *bandwidth* (Siregar, 2009).

Algoritme Hierarchical Token Bucket menggunakan proses penjadwalan seperti dibawah ini untuk pengiriman pesan (Wijaya & Handoko, 2013).

- *Rate* adalah digunakan untuk menentukan *bandwidth* maksimum yang bisa dipakai oleh setiap *class*.
- *Ceil* adalah untuk menentukan peminjaman *bandwidth* antar class yang dilakukan dari *node* paling bawah ke *node* atasnya.

2.5 *Topologi Mesh*

Topologi *Mesh* adalah topologi yang menerapkan hubungan antar komputer, karena setiap komputer bersifat sentral. Pada topologi *mesh* nilai sentral berupa n-1 yaitu ketika terdapat sejumlah n komputer, maka satu komputer akan saling terhubung dengan komputer. Gambar 2.13 menunjukan gambaran ilustrasi kinerja *topologi mesh* (Irianto, 2014). :



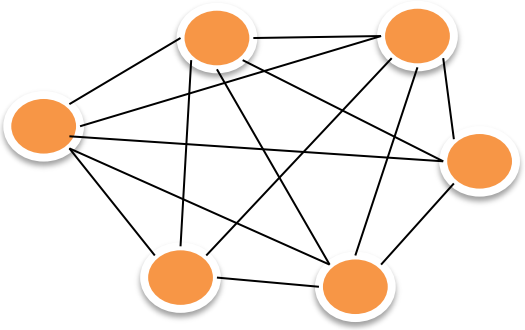
Gambar 2. 11 Topologi *Mesh*

Sumber :[(Irianto, 2014)]

Berdasarkan jumlah dari *node* yang dilalui, *topologi mesh* dibagi menjadi dua bagian yaitu:

1. *Fully Connected*

   Yaitu setiap *node* pada jaringan akan saling terhubung dengan *node* yang lain. Gambar 2.14 menunjukan ilustrasi gambar dari *topologi mesh full connected* (Hariyadi, 2009).



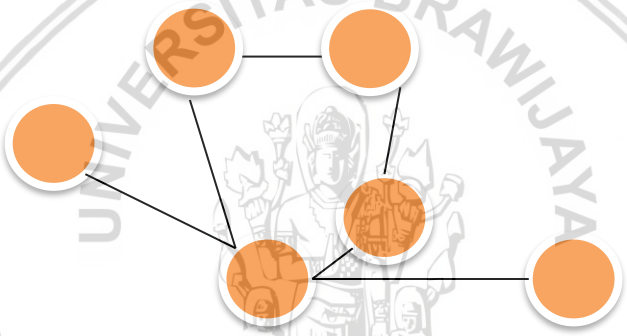
Gambar 2. 12 *Topologi Mesh Full Connected*

Sumber :[(Hariyadi, 2009)]

Pada topologi ini, setiap *node* akan terhubung dengan *node* yang lainnya.

2. *Partially Connected*

Pesan akan dikirimkan dengan mencari jalur terpendek. Gambar 2.15 menunjukan ilustrasi dari gambar *partially connected mesh topology* (Hariyadi, 2009).



Gambar 2. 13 Topologi *Mesh Partially Connected*

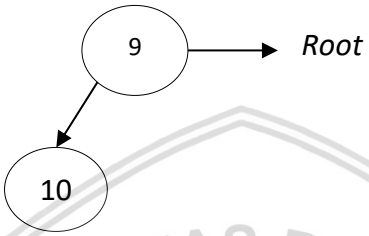
Sumber :[(Hariyadi, 2009)]

Pada topolgi *mesh* dengan tipe *partially connected* tidak semua *node* akan saling terhubung melainkan hanya terhubung secara parsial.

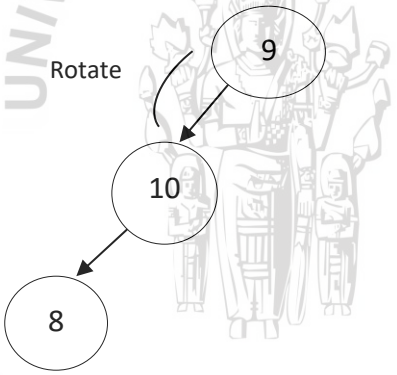
2.6 AVL Tree (Adelson – Velsky - Landis) Tree

Adalah *height balanced 1-tree*. *Height balanced 1-tree* adalah *Binary Search Tree* degan beda tinggi *left subtree* dan *right subtree* pada setiap *node* tidak lebih dari satu level *node*. Penambahan node *AVL tree* sama halnya pada penembahan node *Binary Search Tree* yaitu penelusuran dimulai dari root. Apabila nilai node yang ditambahkan lebih kecil dari nilai node pembanding maka *node* yang ditambahkan menjadi *left child*. Apabila nilai node yang ditambahkan lebih besar dari pada nilai node pembanding, maka yang ditambahkan menjadi *right child*. Setelah node baru dikaitkan pada pohon dilakukan pemeriksaan apakah selisih tinggi pada setiap node tidak lebih dari satu *node*. Gambar 2.16 menunjukan gambar dari *AVL tree* dengan *heigh balanced 1-tree* dengan nilai *node* 10, 9, 8, 7 dan 5, langkah-langkahnya yaitu (Ngoen, 2003).

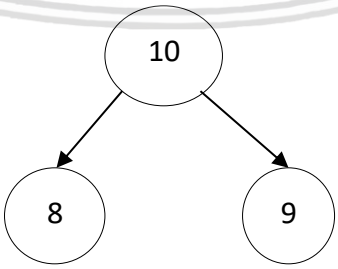
1. Langkah yang pertama yaitu menentukan *node root* terlebih dahulu. Dimana *node root* ditentukan berdasarkan dengan nilai *index node* terbesar yaitu *index* 9. Kemudian akan dilakukan penambahan *node* dengan melakukan pengecekan terlebih dahulu. Jika node yang ditambahkan lebih kecil dari nilai node pembanding maka *node* yang ditambahkan menjadi *left child* dan nilai node yang ditambahkan lebih besar dari pada nilai node pembanding, maka yang ditambahkan menjadi *right child*. Karena *node* baru yang akan ditambahkan lebih kecil dari *node* 9, maka *node baru menjadi left child* dari *node* 9.



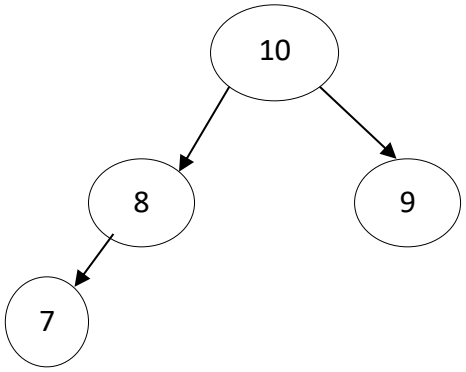
2. Kemudian akan dilakukan penambahan *node* baru yaitu *node* 8. Karena *node* 8 lebih kecil dari *root* dan *node* 10, maka *node* 8 menjadi *left child* dari *node* 10.



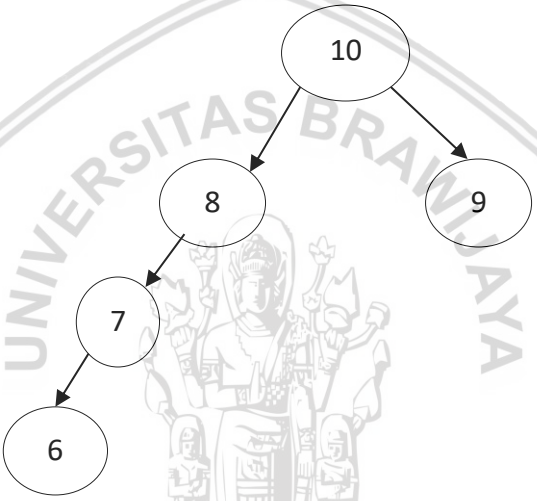
3. Pada *AVL Tree* akan dilakukan *balancing* ketika *left child* dan *right child* melebihi tinggi dari satu *node.*



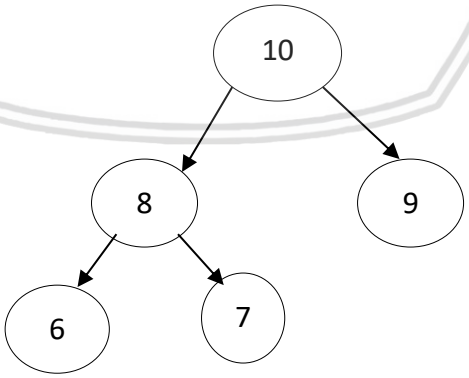
4. Kemudian akan dilakukan penambahan *node* baru yaitu *node* 7. Karena *node* 7 lebih kecil dari *root* dan *node* 8, maka *node* 7 menjadi *left child* dari *node* 8.



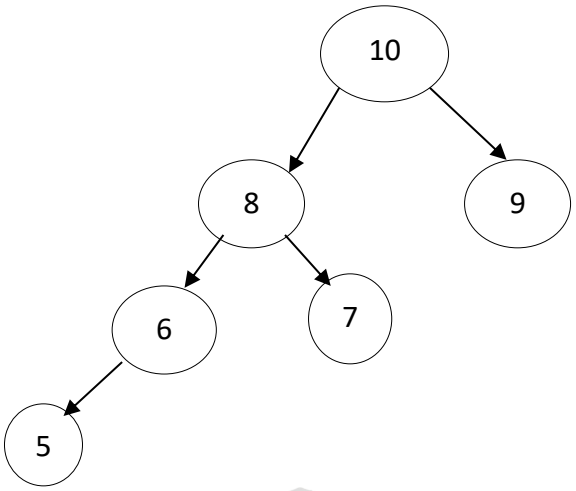
5. Kemudian akan dilakukan penambahan *node* baru yaitu *node* 6. Karena *node* 6 lebih kecil dari *root* dan *node* 7, maka *node* 6 menjadi *left child* dari *node* 7.



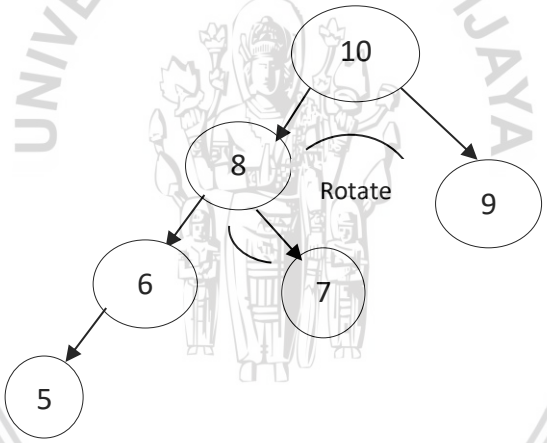
6. Karna *left child* dan *rigth child* memiliki tinggi *node* yang berbeda, maka akan dilakukan *balancing,* seperti:



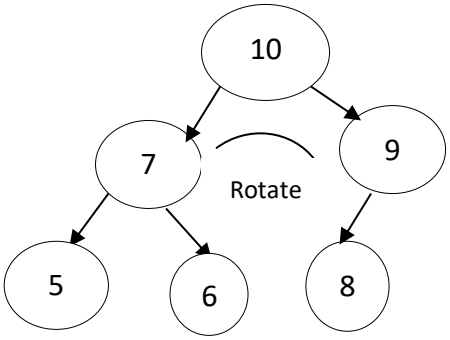
7. Kemudian akan dilakukan penambahan *node* baru yaitu *node* 5. Karena *node* 5 lebih kecil dari *root* dan *node* 6, maka *node* 5 menjadi *left child* dari *node* 6. Dari gambar diatas *left child* dan *right child* tidak seimbang, maka akan dilakukan *balancing,* seperti:



8. Pada sisi *left child* akan dirotasi karena tinggi *node* melebihi 1 *node.* Rotasi akan dilakukan dari *node* 8, karena *node* ini lebih besar di bandingkan dengan *node* 6, 7 dan 5. Ketika *node* 8 dirotasi, maka akan dicek terlebih dahulu apakah *node* 8 lebih kecil atau lebih besar dari *node* pembanding yaitu *node* 9. Karena *node* 8 lebih kecil dari *node* pembanding maka, *node* 8 menjadi *left child* dari *node* 9.



9. Ketika *node* 8 sudah di rotasi, maka *node* 8 akan menjadi *left child* dari *node* 9. Dan *node* 7 akan di rotasi karena lebih besar dari *node* 6 dan *node* 5. Dimana *node* 6 akan menjadi *right child node* 7 dan *node* 5 akan menjadi *left child* dari *node* 7.



Gambar 2. 14 *AVL Tree*
Sumber:[(Ngoen, 2003)]

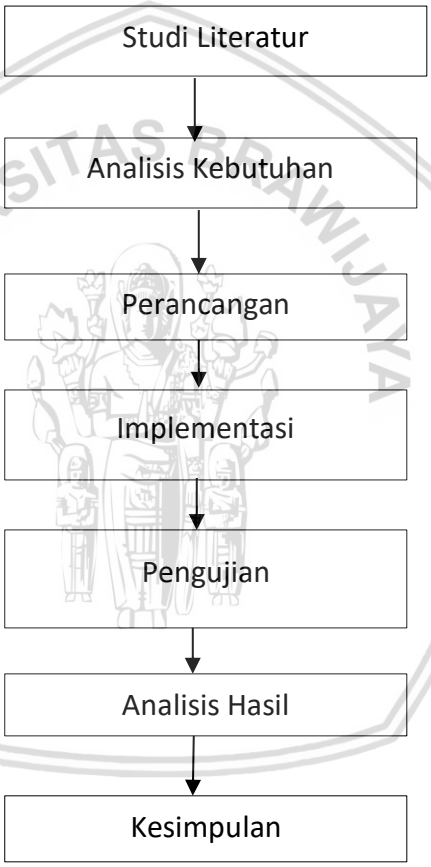
Dari gambar 2.16 merupakan contoh *AVL Tree* dengan *left child* dan *right child* setiap *node* tidak lebih dari satu *node*. Penambahan *node* pada AVL *tree* dimulai dengan penelusuran dari *root*. Jika *node* baru lebih kecil dari nilai *root,* maka *node* baru menjadi *left child* dan jika *node* baru memiliki nilai lebih besar dari *root,* maka menjadi *right child*. Setelah *node* baru dimasukkan pada tree, maka dilakukan pemeriksaan apakah selisih tinggi pada setiap *node* tidak lebih dari satu level. Apabila selisih tinggi lebih dari satu level maka dilakukan *rotasi left child* atau *right child*.

BAB 3 METODOLOGI

Pada bab ini akan dibahas mengenai metode yang digunakan beserta tahap tahap *implementasi Delay Tolerant Network* (DTN) dengan menggunakan *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy*. Adapun tahap metodologi penelitian dan diagram blok metodologi penelitian ditunjukkan pada Gambar 3.1

3.1 Metodologi Penelitian

Pada tahap metodologi penelitian, terdapat langkah-langkah yang digunakan untuk penyelesaian penelitian ini seperti: studi literatur, analisis kebutuhan, perancangan, implementasi, pengujian, analisis hasil dan kesimpulan. Gambar 3.1 menunjukan diagram alir dari pengerjaan penelitian ini yaitu:



Gambar 3. 1 Diagram Blok Metodologi Penelitian

Pada *studi literature* akan berisi informasi yang digunakan untuk pelaksanaan penelitian. Analisis kebutuhan digunakan untuk mengetahui kebutuhan apa saja yang digunakan untuk penelitian seperti perangkat lunak dan perangkat keras. Pada tahap perancangan akan dilakukan proses pembuatan konsep dari *Implementasi protocol Maxprop, Implementasi Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy* dan perancangan topologi. Tahap *implementasi* dilakukan setelah tahap perancangan sudah selesai dan akan masuk pada tahap selanjutnya yaitu tahap pengujian.

Pada tahap pengujian dilakukan untuk mendapatkan hasil yang akan dianalisis pada tahap analisa hasil. Tahap terakhir adalah kesimpulan. Kesimpulan dibuat untuk mengetahui hasil dari *protocol MaxProp* dan *Algoritme Hierarchical Token Bucket*.

3.2 Studi Literatur

Studi literatur pada penelitian ini adalah mempelajari literatur dari beberapa informasi dan pustaka yang berkaitan dengan penelitian yang akan dilakukan. Studi literatur diperoleh dari buku, jurnal, *e-book* dan bimbingan oleh dosen pembimbing. Adapun literatur yang dipelajari yaitu:

a. *DTN (Delay Tolerant Network)*
   Pada *studi* literatur *delay tolerant network* memepelajari hal – hal yang berkaitan dengan *Delay Tolerent Network* seperti apa itu *DTN,* karakteristik *DTN*, cara kerja *DTN* dan klasifikasi *protocol routing* dalam *DTN*.
b. Seleksi *Node*
   Pada *studi* literatur ini mempelajari tentang seleksi *node* yang akan digunakan untuk proses implementasi *Algoritme Hierarchical Token Bucket.* Seleksi *node* dilakukan dilakukan menggunakan mekanisme *two-hop routing.*
c. *Algoritme Hierarchical Token Bucket*
   Pada *studi* literatur ini mempelajari tentang *Algoritme Hierarchical Token Bucket* seperti apa itu *Algoritme Hierarchical Token Bucket,* karakteristik serta proses peminjaman *bandwidth* yang pertama dan yang kedua pada *Algoritme Hierarchical Token Bucket.*
d. *Topologi Mesh*
   Pada *studi* literatur ini mempelajari tentang *topologi mesh* yang digunakan sebagai tempat komunikasi antar *node* dalam implementasi *Delay Tolerant Network* menggunakan *Algoritme Hierarchical Token Bucket*.
e. *AVL Tree*
   Pada *studi* literatur ini mempelajari tentang *AVL Tree. AVL Tree* digunakan sebagai struktur data untuk pembuatan *node* pada *Algoritme Hierarchical Token Bucket.*
f. *Bundle Layer*
   Pada *studi* literatur ini mempelajari tentang *bundle layer.* Dimana *bundle layer* ini digunakan untuk menyimpan pesan sementara pada *Delay Tolerant Network* yang berada diatas *layer transport.*

3.3 Analisis Kebutuhan

Analisis kebutuhan dilakukan untuk mengetahui kebutuhan apa saja yang diperlukan dalam proses Implementasi *Delay Tolerant Network (DTN)* Dengan Menggunakan *Algoritme Hierarchical Token Bucket* Untuk Seleksi *Node Routing Multi copy*. Pada tahap analisis kebutuhan dibagi menjadi analisis kebutuhan perangkat keras dan analisis kebutuhan perangkat lunak. Analisis kebutuhan perangkat keras berisi informasi apa saja yang digunakan dalam proses simulasi yang akan digunakan. Sedangkan, analisis kebutuhan perangkat lunak berisi tentang *tools* apa saja yang digunakan dalam penelitian ini.

3.4 Perancangan

Tahap perancangan dibuat untuk membuat langkah kerja dari *protocol MaxProp* dan *Algoritme Hierarchical Token Bucket* secara menyeluruh agar mencapai tujuan dari penelitian. Hasil dari tahap perancangan, akan digunakan untuk proses berikutnya yaitu proses *implementasi* dan pengujian. Pada tahap ini terdapat tiga model perancangan yaitu perancangan untuk *protocol MaxProp*, perancangan *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy* dan perancangan topologi. Perancangan topologi adalah perancangan tentang gambaran umum *topologi* penelitian yang diterapkan untuk mengetahui bagaimana sistem berjalan dan skenario penelitian yang akan digunakan dan untuk mengetahui koneksi antar *node*. Dari hasil perancangan *protocol MaxProp*, akan dianalisis dengan menggunakan 5 parameter uji yaitu Jumlah *node*, *routing protocol*, ukuran pesan, waktu simulasi dan kecepatan *node*.

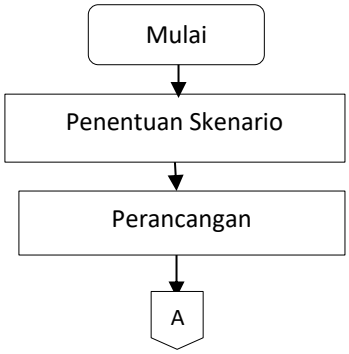
Tabel 3. 1 Parameter Simulasi

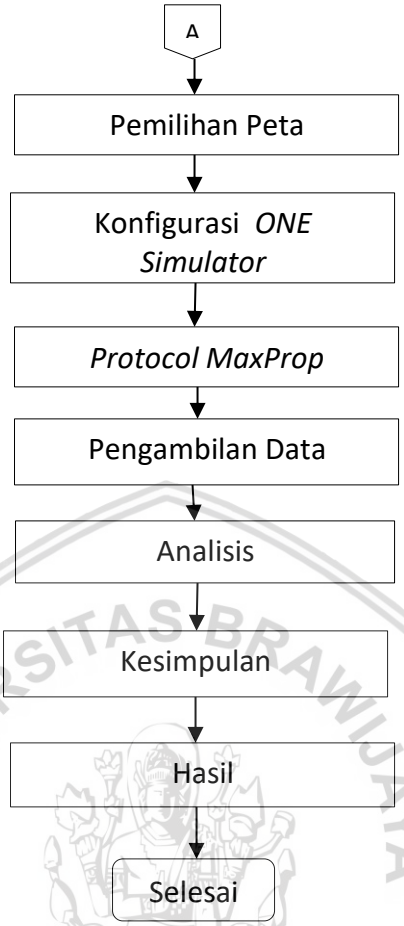
Parameter	Nilai
Jumlah *Node*	50, 100, 150 dan 200 *node*
Routing Protocol	*MaxProp*
Waktu simulasi	3600s
Ukuran pesan	1 MB
Kecepatan *Node*	(20-40), (40-80), (80-160) in km/jam

Jumlah *node* yang digunakan yaitu 50, 100, 150 *node* dan 200 *node*. Jumlah ini dipilih berdasarkan jumlah maksimal pengunjung pada daerah simulasi Jalur Lintas Selatan Malang untuk 1 minggu. Sedangkan untuk *protocol* yang digunakan yaitu *protocol MaxProp*. Waktu simulasi yang digunakan yaitu 3600s. Ukuran pesan yang digunakan yaitu 1MB. Kecepatan *node* yang digunakan yaitu 20-40, 40-80 dan 80-160 km/jam. Kecepatan *node* yang berbeda-beda ditujukan untuk mengetahui pengaruh penambahan kecepatan *node* pada *protocol MaxProp*.

3.4.1 Perancangan Protokol *MaxProp*

Pada perancangan ini dilakukan untuk mnegetahui apa saja yang dibutuhkan untuk proses implementasi yang akan dilakukan seperti penentuan skenario, pemilihan peta dan konfigurasi pada *folder ONE Simulator.* Gambar 3.2 menunjukan *flow chart* dari *implementasi protocol MaxProp,* yaitu:



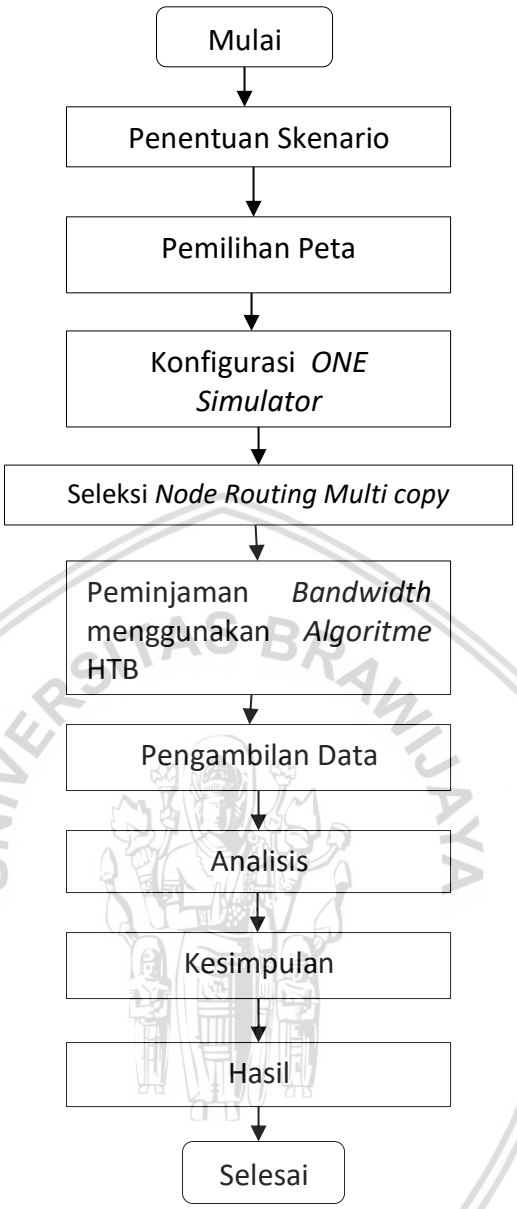


Gambar 3. 2 *Flow Chart* Umum *Protocol MaxProp*

Pada perancangan *protocol MaxProp* tahap pertama dimulai dengan menentukan skenario yaitu menentukan alur dari pengujian *protocol MaxProp* yang akan dilakukan. Kemudian dilakukan perancangan yang terdiri dari pemilihan peta dan pembuatan peta menggunakan *tools* bantuan yaitu *OPENJumps.* Perancangan peta bertujuan untuk menentukan daerah simulasi yang akan digunakan. Selanjutnya akan dilakukan konfigurasi pada *file default setting ONE Simulator* yang mengacu pada tabel 3.3. setelah proses konfigurasi *ONE Simulator* sudah dilakukan, akan dilakukan proses pengambilan data berdasarkan nilai parameter pengujian yaitu *average latency, overhead ratio, delivery probability* dan *average hop count.* Jika nilai dari parameter pengujian sudah didapat, maka dilakukan proses analisis untuk mengetahui kinerja dari setiap parameter pengujian pada *protocol MaxProp,* kemudian dilakukan proses kesimpulan untuk mendapatkan hasil sesuai dengan simulasi yang dilakukan.

3.4.2 Perancangan *Algoritme Hierarchical Token Bucket*

Pada tahap ini, dilakukan perancangan untuk *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy.* Gambar 3.3 menunjukan gambaran umum dari *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy* yaitu:



Gambar 3. 3 Perancangan Umum *Algoritme Hierarchical Token Bucket* Untuk Seleksi *Node Routing Multi Copy*

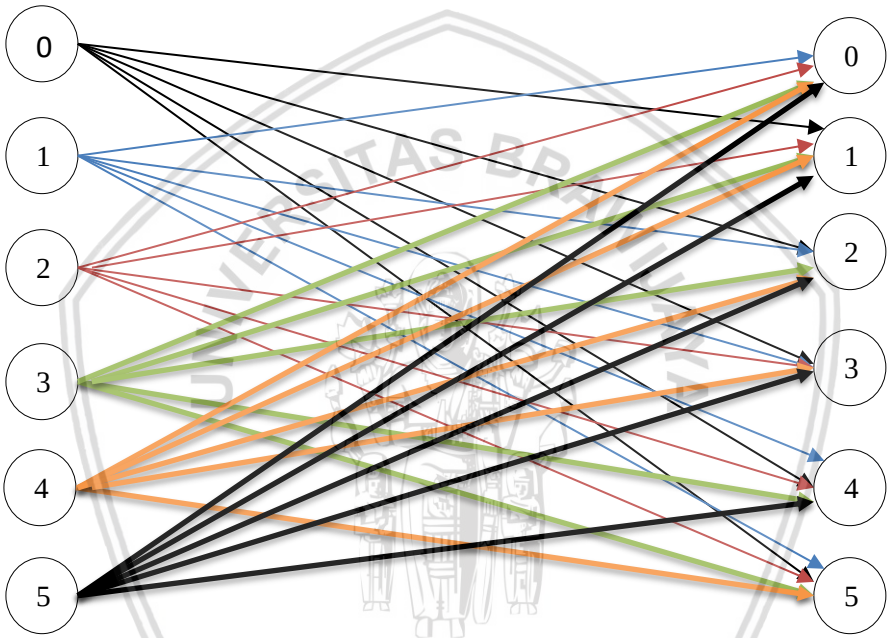
Pada perancangan *Algoritme Hierarchical Token Bucket* dilakukan dengan seleksi *node routing multi copy.* Tahap pertama dimulai dengan menentukan skenario yaitu menentukan alur dari pengujian *protocol MaxProp* yang akan dilakukan. Kemudian dilakukan perancangan yang terdiri dari pemilihan peta dan pembuatan peta menggunakan *tools* bantuan yaitu *OPENJumps.* Selanjutnya akan dilakukan konfigurasi pada *file default setting ONE Simulator* untuk melakukan konfigurasi yang mengacu pada tabel 3.3 dan akan dilakukan proses seleksi *node routing multi copy.* Kemudian akan dilakukan pengambilan data dari hasil seleksi *node routing multi copy* yang akan digunakan sebagai peminjaman *bandwidth* pada *Algoritme Hierarchical Token Bucket.* Jika nilai dari parameter pengujian sudah didapat, maka dilakukan proses analisis untuk mengetahui kinerja dari setiap parameter pengujian yang diterapkan pada *protocol MaxProp*

kemudian dilakukan proses kesimpulan untuk mendapatkan hasil sesuai dengan simulasi yang dilakukan.

3.4.3 Perancangan Topologi

Pada tahap ini dilakukan perancangan topologi yang digunakan untuk komunikasi antar *node.* Topologi *Mesh* adalah topologi yang menerapkan hubungan antar komputer, karena setiap komputer bersifat sentral berupa n-1/2 yaitu ketika terdapat sejumlah n komputer, maka satu komputer akan saling terhubung dengan komputer yang lain untuk nilai *port* I/O masing-masing komputer dan n-1/2 untuk pembangunan jaringan *topologi mesh*. Gambar 3.4 menunjukan ilustrasi dari hubungan antar *node* pada topologi *mesh,* yaitu:

Gambar 3. 4 Hubungan Antar *Node* Pada Topologi *Mesh*



Keterangan:

Pada topologi *mesh, node* akan saling terhubung dengan *node* yang lain dengan nilai n-1, n adalah jumlah *node.* Dari gambar 3.4 diperoleh hubungan untuk setiap *node* nya yaitu:

- *Node* 0 terhubung dengan *node* 1, 2, 3, 4 dan 5.
- *Node* 1 terhubung dengan *node* 2, 3, 4, 5 dan 0.
- *Node* 2 terhubung dengan *node* 3, 4, 5, 0 dan 1.
- *Node* 3 terhubung dengan *node* 4, 5, 0, 1 dan 2.
- *Node 4* terhubung dengan *node* 5, 0, 1, 2 dan 3.
- *Node* 5 terhubung dengan *node* 0, 1, 2, 3 dan 4.

Topologi mesh dipilih karena sesuai dengan konsep pada penelitian yang akan dilakukan yaitu dengan melakukan seleksi *node routing multi copy* menggunakana *Algoritme Hierarchical Token Bucket*. Seleksi *node* dilakukan dengan *two hop routing* yaitu pesan hanya akan dikirimkan pada *node* yang menjadi tujuan *node* sumber.

3.5 Implementasi

Pada tahap ini akan dijelaskan bagaimana proses dari implementasi kebutuhan dan perancangan pada penelitian yang akan dilakukan. Implementasi yang akan dilakukan antara lain:

1. *Implementasi Delay Tolerant Network* dengan menggunakan *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy.*

   *Implemen*tasi pada tahap ini mencakup seluruh proses sistem dengan melakukan seleksi *node* terlebih dahulu pada *protocol MaxProp* dengan mekanisme *two hop routing* untuk kemudian diterapkan pada *Algoritme Hierarchical Token Bucket.*
2. *Implementasi protocol Maxprop*.

   Implementasi pada tahap ini dilakukan dengan melakukan simulasi langsung pada *protocol MaxProp*.

3.6 Pengujian

Pengujian akan dilakukan dengan mensimulasikan *protocol maxprop* dan *Algoritme Hierarchical Token Bucket* untuk seleksi *node* pada *routing multi copy*. Tujuan dilakukan pengujian adalah untuk melakukan penilaian dan evaluasi mekanisme pada *protocol MaxProp* dan *Algoritme Hierarchical Token Bucket* pada JLS(Jalur Lintas Selatan) Malang. Parameter uji yang digunakan adalah Jumlah *node*, *routing protocol*, ukuran pesan, waktu simulasi dan kecepatan *node*. Dari parameter uji tersebut akan dilakukan proses analisis untuk mendapatkan nilai *overhead ratio, average latency, delivery probability* dan *average hop count* pada *protocol MaxProp* dan *Algoritme Hierarchical Token Bucket*. Berikut daftar parameter perancangan untuk *protocol MaxProp* dan *Algoritme Hierarchical Token Bucket*.

Tabel 3. 2 Parameter Simulasi *Protocol MaxProp* dan *Algoritme Hierarchical Token Bucket*

Parameter	Nilai
Jumlah *Node*	50, 100, 150 dan 200 *node*
Routing Protocol	*MaxProp*
Waktu simulasi	3600s
Ukuran pesan	1 MB
Kecepatan *Node*	(20-40), (40-80), (80-160) in km/jam

Jumlah *node* yang digunakan yaitu 50, 100, 150 *node* dan 200 *node*. Jumlah ini dipilih berdasarkan jumlah maksimal pengunjung pada daerah simulasi Jalur Lintas Selatan Malang untuk 1 minggu. Sedangkan untuk *protocol* yang digunakan yaitu *protocol MaxProp. Protocol* ini dipilih karena, cara kerja pengiriman pesan yang memiliki nilai sama dengan mekanisme *two hop rouitng* yaitu bernilai n+1 setiap melakukan pengiriman pesan. Waktu simulasi yang digunakan yaitu 3600s. Ukuran pesan yang digunakan yaitu 1MB. Kecepatan *node* yang digunakan yaitu 20-40, 40-80 dan 80-160 km/jam. Kecepatan *node* yang berbeda-beda ditujukan untuk mengetahui seberapa banyak pesan yang terkirim berdasarkan jumlah *node* yang sama pada *protocol MaxProp*.

Setelah melakukan konfigurasi pada *file default setting* pada *ONE Simulator* yang mengacu pada tabel 3.3, maka akan dilakukan proses analisis hasil dari hasil *Implementasi Delay Tolerant Network* dengan menggunakan *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy* atau *Implementasi protocol Maxprop* menggunakan beberapa parameter performansi. Tabel 3.3 menunjukan parameter performansi yang akan digunakan dalam penelitian ini, yaitu:

Tabel 3. 3 Parameter Pengujian

Parameter Performansi
Overhead Ratio
Average Latency
Delivery Probability
Average Hop Count

Hasil dari metrik performansi akan digunakan untuk analisis terhadap *protocol MaxProp* dan *Algoritme Hierarchical Token Bucket* untuk mendapatkan kesimpulan dari kedua pengujian. Berikut penjelasan dari parameter pengujian yang akan digunakan dalan pengujian ini, seperti:

1. Average Latency

Adalah rata-rata waktu ketika pesan dikirim dan diterima oleh *node* tujuan (Bindra & Sangal, 2012).

2. Overhead Ratio

Parameter yang digunakan untuk mengetahui berapa banyak *relay node* yang terkirim dari sumber ke tujuan. Parameter ini dipengaruhi oleh waktu yang digunakan oleh pesan dalam *buffer* sebelum pesan diteruskan ke *node* yang lain. Semakin rendah nilai *overhead ratio,* maka semakin optimal juga *protocol routing* yang digunakan. (Alaoui, et al., 2015).

$$\boldsymbol{Overhead\ Ratio}\ \frac{(R-D)}{D} \qquad \textbf{(3.1)}$$

Keterangan:

R: Jumlah pesan yang sampai pada tujuan

D: Jumlah pesan yang diteruskan oleh *node*

3. Delivery Probability

Perbandingan antara banyak nya total pesan yang dikirim pada *node* tujuan dengan jumlah pesan yang dibuat oleh *node* sumber yang akan dikirimkan dalam waktu yang ditentukan. Nilai *delivery probability* yang tinggi menentukan seberapa banyak pesan yang sampai pada *node* tujuan (Kumar, et al., 2015).

$$\boldsymbol{Delivery\ Probability}\ \frac{D}{G} \qquad \textbf{(3.2)}$$

Keterangan:

D: jumlah pesan yang sampai pada tujuan

G: jumlah pesan yang dibuat oleh *node* sumber

4. Average Hop Count

Adalah jumlah rata-rata pesan yang diolah untuk sampai pada tujuan (Patel & Gondaliya, 2015).

3.7 Analisis Hasil

Hasil dari proses pengujian akan digunakan untuk proses analisis. Analisis hasil dilakukan dengan membandingkan hasil parameter pengujian antara *implementasi protocol MaxProp* dan *implementasi Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy* yang mengacu pada tabel 3.3.

3.8 Kesimpulan

Kesimpulan bisa didapat setelah semua tahapan pada perancangan, *implementasi,* pengujian dan analisis hasil sudah dilakukan. Pada kesimpulan, terdapat saran yang ditujukan untuk pengembangan *protocol routing* yang lain, penggunaan *Algoritme* yang lain dan penerapan mekanisme seleksi *node* yang lain.

BAB 4 REKAYASA KEBUTUHAN

Pada tahap ini akan dijelaskan mengenai deskripsi umum mengenai sistem yang akan dibuat dan analisis kebutuhan sistem yang terdiri dari analisis kebutuhan perangkat keras dan analisis kebutuhan perangkat lunak.

4.1 Depskripsi Umum Sistem

Hierarchical Token Bucket adalah sistem yang digunakan untuk manajemen *bandwidth*. Dimana jumlah *bandwidth* yang diminta tidak boleh melebihi dari kapasitas *bandwidth* yang disediakan oleh *sistem*. *Hierarchical Token Bucket* berperan dalam mengontrol penggunaan *bandwidth* terhadap *link* yang diberikan kepada *client* dan membatasi *download* dan *upload clien*. Untuk menerapkan *Algoritme* ini diperlukan pembuatan *node* menggunakan *AVL Tree. Node* yang sudah dibuat akan berkomunikasi melalui topologi *mesh.* Ketika pembuatan topologi sudah berhasil, maka komunikasi antar *node* bisa dilakukan untuk melakukan pengiriman pesan. Sebelum *node* sumber mengirim pesan, dilakukan seleksi *node* terlebih dahulu meggunakan mekanisme *two hop routing.* Dimana pesan hanya akan dikirimkan pada *node* yang menjadi tujuan dari *node* sumber yang dikategorikan berdasarkan *index node* ganjil dan *node* genap.

Sistem ini akan diterapkan pada arsitektur *Delay Tolerant Network,* sehingga dibutuhkan sebuah lingkungan sistem yang mendukung arsitektur tersebut. Oleh karena itu, penelitian ini menggunakan simulasi *ONE Simulator*. Karena *Algoritme Hierarchical Token Bucket* diterapkan pada *Delay Tolerant Network*, maka dibutuhkan subsistem yang bekerja sama untuk membentuk sistem yang utuh yang terdiri dari subsistem *server*, subsistem *intermediatte node*, dan subsistem *client*. Berikut ini penjelasan dari masing-masing subsistem:

- Subsistem *server* pada subsistem berperan sebagai penyedia *bandwidth* atau server yang digunakan untuk meminta sejumlah *bandwidth* yang dilakukan oleh *node.*
- Subsistem *intermediatte node*, pada subsistem *intermediatte node* berperan sebagai *gateway* yaitu penghubung antara *subsistem server* dan subsistem *client* untuk proses pengiriman dan penerimaan pesan.
- Subsistem *client*, pada subsistem *client* berperan sebagai *node* yang melakukan peminjaman *bandwidth.* terdapat perbedaan antara *intermediatte node* dan *client* yaitu akan dilakukan proses *delete bundle* atau *save bundle* setelah dilakukan *duplicate bundle* pada pihak *client.* Sedagkan persamaan nya yaitu sama-sama menerima *bundle* yang dikirim oleh pihak *server.*

4.2 Analisis Kebutuhan

Analisis kebutuhan dilakukan untuk mengetahui kebutuhan apa saja yang diperlukan dalam proses Implementasi *Delay Tolerant Network (DTN)* Dengan Menggunakan *Algoritme Hierarchical Token Bucket* Untuk Seleksi *Node Routing Multi copy*. Pada tahap analisis kebutuhan dibagi menjadi analisis kebutuhan perangkat keras dan analisis kebutuhan perangkat lunak.

4.2.1 Analisis Kebutuhan Perangkat Keras

1. 1 buah komputer dengan spesifikasi sebagai berikut:

Tabel 4. 1 Kebutuhan Perangkat Keras

No	Kebutuhan Perangkat Keras	Spesifikasi
1	*Processor*	*Intel Celerone* N3050, 1.60GHz
2	RAM	2 GB
3	*Hardisk*	500 GB
4	Monitor	14inch

Pada analisis kebutuhan perangkat keras berisi informasi apa saja yang digunakan dalan proses implementasi *Delay Tolerant Network* menggunakan *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy* seperti 1 buah komputer dengan spesifikasi *processor* berupa *Intel Celerone* N3050, 1.60GHz dengan kapasitas RAM 2GB, *hardisk* 500GB dengan monitor 14inch.

4.2.2 Analisis Kebutuhan Perangkat Lunak

Tabel 4. 2 Kebutuhan Perangkat Lunak

No	Kebutuhan Perangkat Lunak	Spesifikasi
1	*Operating System*	*Windows 10*
2	Simulator	*The ONE Simulator* 1.4.1 RC
3	Bahasa Pemrograman	JAVA
4	*Tools* untuk pembuatan peta	*OpenJUMP* 1.1.0
5	*Tools* untuk merubah peta asli	*OSM2 WKT*

Pada analisis kebutuhan perangkat lunak berisi *tools* apa saja yang digunakan dalan proses implementasi *Delay Tolerant Network* menggunakan *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy. Tools* yang dibutuhkan yaitu *OS Windows 10* yang digunakan sebagai *tools* pembuatan dokumentasi dan proses implementasi *Delay Tolerant Network* menggunakan *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy.* Pada penelitian ini menggunakan *ONE Simulator* sebagai simulasi proses pengujian. Untuk proses implentasi *Algoritme Hierarchical Token Bucket* menggunakan bahasa pemrograman *Java,* karena bahasa pemrograman ini memiliki basis yang sama pada *tools* pembuatan peta yang bersifat *Java*. Sedangkan pada proses pembuatan peta menggunakan *tools OpenJUMP* 1.1.0 dimana peta yang dibuat dengan ekstensi .png atau .jpg yang kemudian di ekstrak menggunakan *tools OSM2 WKT* atau dengan membuat manual pada *OpenJUMP* 1.1.0.

BAB 5 PERANCANGAN DAN IMPLEMENTASI

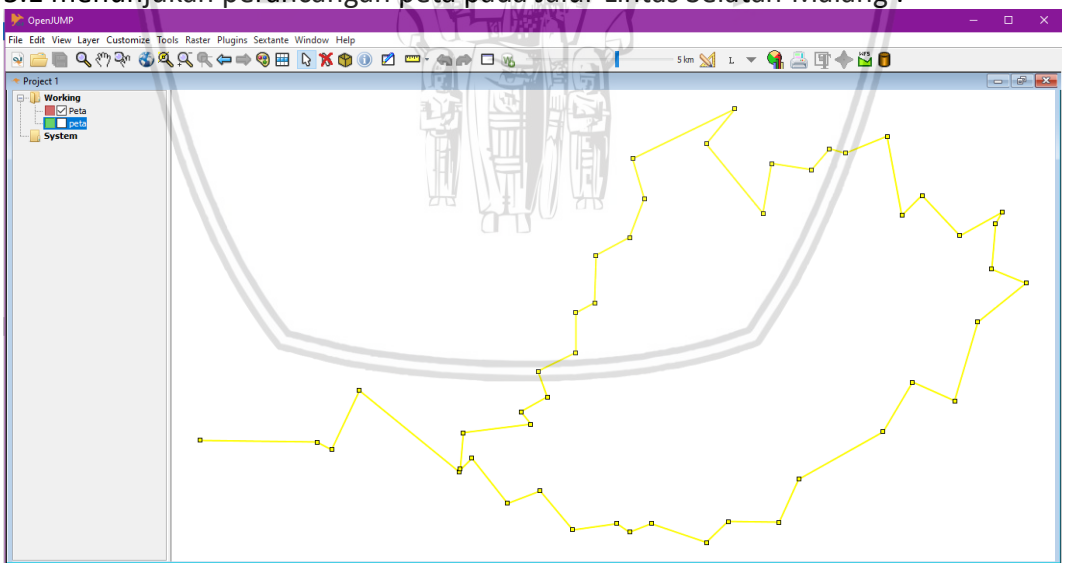
Pada tahap ini akan dijelaskan mengenai perancangan dan implementasi dari penelitian yang akan dilakukan. Terdapat beberapa tahapan pada perancangan yaitu perancangan peta, perancangan *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy, protocol MaxProp* dan perancangan sistem. Dan beberapa tahapan pada implementasi yaitu implementasi *protocol MaxProp* dan implementasi *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy.*

5.1 Perancangan Skema Penelitian

Pada tahap ini akan didefinisikan mengenai peta yang akan digunakan dan perancangan untuk *protocol MaxProp* tanpa menggunakan *Algoritme Hierarchical Token Bucket* dan *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy*.

5.1.1 Perancangan Peta

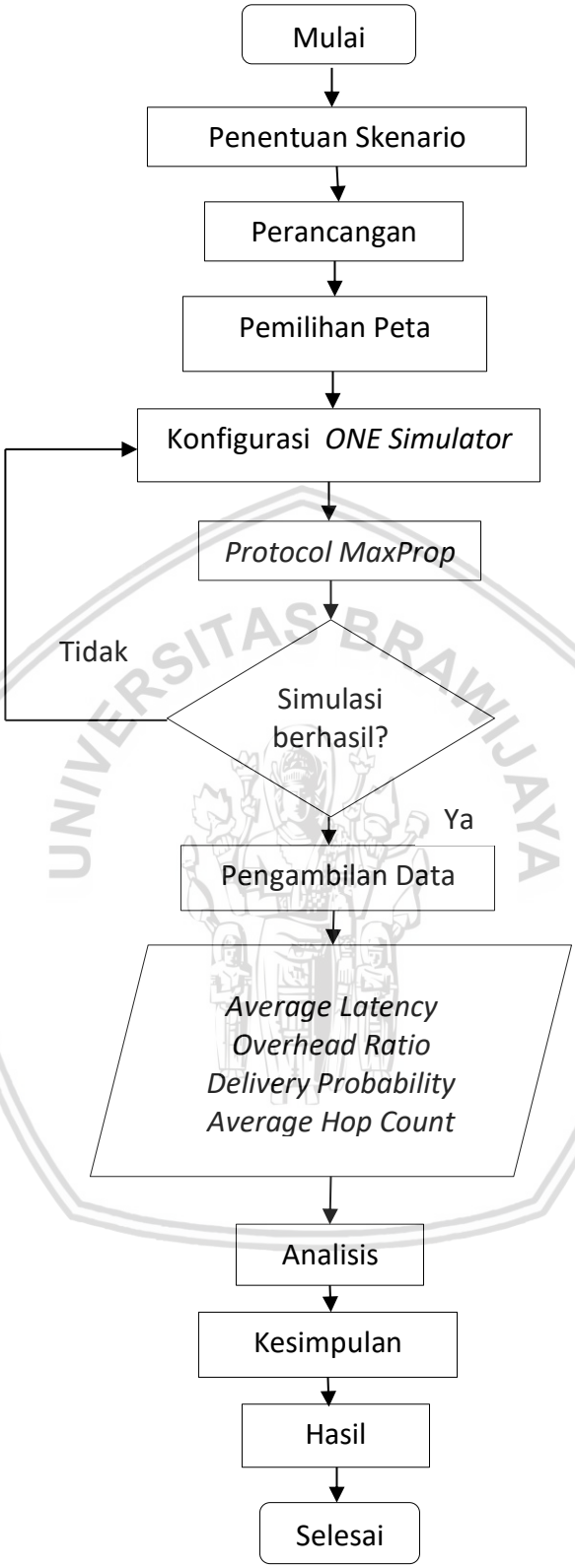
Perancangan peta ditujukan untuk pembuatan peta penelitian dengan menggunakan Jalur Lintas Selatan (JLS) Malang sebagai tempat simulasi dengan menggunakan *tools* bantuan yaitu *OPENJums*. Hasil dari pembuatan peta disimpan kedalam format *file .wkt (well known text)* karena, *ONE Simulator* hanya dapat membanca *file* dengan *format .wkt.* Pada tahap ini akan diuraikan mengenai skenario dari *protocol* dan *Algoritme* yang akan digunakan. Gambar 5.1 menunjukan perancangan peta pada Jalur Lintas Selatan Malang :



Gambar 5. 1 Peta Jalur Lintas Selatan Malang

5.1.1.1 Skenario *Protocol MaxProp*

Pada skenario ini, pengujian akan dilakukan tanpa menggunakan *Algoritme Hierarchical Token Bucket.* Pengujian akan dilakukan setelah melakukan beberapa konfigurasi pada *file default_setting* pada *ONE Simulator.* Gambar 5.2 menunjukan *flowchart* untuk pengujian *protocol MaxProp,* yaitu:



Gambar 5. 2 *Flowchar Protocol MaxProp*

Tahap pertama dimulai dengan menentukan skenario yaitu menentukan alur dari pengujian *protocol MaxProp* yang akan dilakukan. Kemudian dilakukan perancangan yang terdiri dari pemilihan peta dan pembuatan peta menggunakan

tools bantuan yaitu *OPENJumps.* Selanjutnya akan dilakukan konfigurasi *file default setting ONE Simulator* yang mengacu pada tabel 3.3. Jika simulasi gagal, maka kembali pada proses konfigurasi *file ONE Simulator* dengan menggunakan *protocol MaxProp* sebagai *router.* Pengambilan data akan dilakukan setelah simulasi yang dilakukan berhasil berdasarkan nilai parameter pengujian yaitu *average latency, overhead ratio, delivery probabiliy* dan *average hop count.* Jika nilai dari parameter pengujian sudah didapat, maka dilakukan proses analisis untuk mengetahui kinerja dari setiap parameter pengujian yang diterapkan pada *protocol MaxProp* kemudian dilakukan proses kesimpulan untuk mendapatkan hasil sesuai dengan simulasi yang dilakukan. Tabel 5.1 menunjukan skenario untuk *protocol MaxProp* dan *Algoritme Hierarchical Token Bucket* yang digunakan:

Tabel 5. 1 Konfigurasi *file ONE Simulator*

No	*Skenario*	*Protocol MaxProp dan Algoritme Hierarchical Token Bucket*
1	*Scenario.name*	*Simulation21*
2	*Scenario.simulateConnections*	*True*
3	*Scenario.updateInterval*	0.1
4	*Scenario.endTime*	3600s
5	*Wifi80211b.type*	*SimpleBroadcastInterface*
6	*Wifi80211b.transmitSpeed*	1375k
7	*Wifi80211b.transmitRange*	100m
8	*highspeedInterface.type*	*SimpleBroadcastInterface*
9	*highspeedInterface.transmitSpeed*	10M
10	*highspeedInterface.transmitRange*	100
11	*Scenario.nrofHostGroups*	1
12	*Group.movementModel*	*MapRouteMovement*
13	*Group.router*	*MaxPropRouter*
14	*Group.bufferSize*	1GB
15	*Group.transmitRange*	10
16	*Group.transmitSpeed*	250k
17	*Group.waitTime*	0,120
18	*Group.nrofInterface*	1
19	*Group.speed*	(20-40), (40-80) dan (80-160)km/jam
20	*Group.nrofHosts*	150
21	*Group1.groupID*	P
22	*Group1.routeFile*	*Map/Peta.wkt*
23	*Group1.routeType*	1
24	*Group1.waitTime*	0,120
25	*Events1.hosts*	0,149
26	*Events1.prefix*	M
27	*MovementModel.rngSeed*	1
28	*MovementModel.worldSize*	5000,5000
29	*MovementModel.warmup*	1000
30	*MapBasedMovement.nrofMapFiles*	1
31	*MapBasedMovement.mapFile1*	*Map/Peta.wkt*
32	*Report.nrofReports*	1
33	*Report.warmup*	0
34	*Report.reportDir*	*Reports/*
35	*Report.report1*	*MessageStatsReport*
36	*ProphetRouter.secondsInTimeUnit*	30
37	*SprayAndWaitRouter.nrofCopies*	6
38	*SprayAndWaitRouter.binaryMode*	*True*
39	*Optimization.connectionAlg*	2

40	*Optimization.cellSizeMult*	5
41	*Optimization.randomizeUpdateOrder*	*True*
42	*GUI.UnderlayImage.fileName*	*data/helsinki_underlay.png*
43	*GUI.UnderlayImage.offset*	64,20
44	*GUI.UnderlayImage.scale*	4.75
45	*GUI.UnderlayImage.rotate*	-0.015
46	*GUI.EventLogPanel.nrofEvents*	30
47	*Events.nrof*	1
48	*Events1.interval*	25.35
49	*Events1.size*	1MB
50	*Events1.class*	*MessageEventGenerator*

Sumber : [Perancangan]

Keterangan:

1. *Scenario.name* digunakan untuk penamaan pada saat simulasi berlangsung yang harus dirubah setiap kali dilakukannya simulasi. Hal ini ditujukan agar laporan yang dihasilkan untuk setiap simulasi akan berganti dengan konten yang baru dan *file* laporan yang terpisah untuk setiap simulasi yang dilakukan.

   #Konfigurasi *Protocol routing* yang digunakan untuk simulasi
   Scenario.name = [simulation21]

2. *Scenario.simulateConnections* harus dalam kondisi *true*, jika *false* maka tidak lagi digunakan untuk menjalankan simulasi melainkan untuk proses *download* pada *CRAWDAD.*

   Scenario.simulaeConnections = true

3. *Scenario.updateInterval* digunakan untuk mendefinisikan *interval* fungsi *update()* dalam *file MessageRouter.java* dengan nilai 0.1s.

   Scenario.updateInterval = 0.1

4. *Scenario.endTime* digunakan untuk konfigurasi waktu simulasi yang akan dijalankan dengan format waktu detik. Dalam penelitian ini waktu simulasi yang digunakan yaitu 1 jam atau 3600s.

   #Konfigurasi waktu simulasi yang digunakan
   #1 jam = 3600s
   Scenario.endTime = 3600

5. *Wifi80211.type* adalah *interface* yang digunakan oleh setiap *node.* Pada penelitian ini *interface* yang digunakan yaitu *wifi802.11b.*

   #jenis *interface* yang digunakan oleh setiap *node*
   #cakupan area lebih luas dan kecepatan transmisi lebih cepat dari pada wifi 802.11a
   #"wifi80211b" *interface for all node*
   Wifi80211b.type = SimpleBroadcastInterface

6. *Wifi80211.transmitSpeed* mendefinisikan kecepatan dari *interface* yang digunakan oleh setiap *node. Wifi802.11b* memiliki kecepatan transmisi minimum 3Mbps atau 375k dan kecepatan transmisi maksimum 11Mbps atau 1375k. Pada penelitian ini kecepatan *interface* yang digunakan yaitu 11 MB atau 1375Kbps. Kecepetan ini adalah kecepatan maksimum yang digunakan oleh *wifi802.11b.*

   #Kecepatan transmisi antarmuka pada *wifi 802.11b*

#*Transmit speed of* 11 Mbps = 1375kBps *Wifi80211b.transmitSpeed = 1375k*

7. *Wifi80211.transmitRange* mendefinisikan cakupan area pada *interface* yang digunakan oleh setiap *node.* Pada penelitian ini *range* yang digunakan yaitu 100m.

Wifi80211b.transmitRange = 100

8. *Scenario.nrofHostGroups* mendefinisikan jumlah *host* yang digunakan oleh group yang disesuaikan oleh kebutuhan, seperti pada penelitian ini hanya menggunakan 1 *host* yaitu p.

#nomer *host* dengan menggunakan 1 *host* yaitu p *Scenario.nrofHostGroups = 1*

9. *Group.movementModel* mendefinisikan *mobility model* yang digunaka dalam simulasi. Pada penelitian ini *mobility model* yang digunakan yaitu *MapRouteMovement* yaitu *mobility model* yang hanya mengikuti jalur yang sudah didefinisikan pada peta.

Group.movementModel = MapRouteMovement

10. *Group.router* mendefinisikan *router* yang digunakan dalam proses simulasi yaitu *MaxPropRouter.*

Group.router = MaxPropRouter

11. *Group.bufferSize* mendefinisikan ukuran *buffer default* untuk semua *node* yaitu 1GB yang digunakan untuk *group* yang sudah didefinisikan.

Group.bufferSize = 1G

12. *Group.transmitRange* mendefinisikan cakupan area transmisi yang digunakan oleh *group.*

Group.transmitRange = 10

13. Group.transmitSpeed mendefinisikan kecepatan transmisi yang digunakan oleh *group* yang sudah didefinisikan dengan kecepatan 2Mbps.

#transmit speed of 2 Mbps = 250kBps *Group.transmitSpeed = 250k*

14. *Group.waitTime* mendefinisikan waktu tunggu minimal dan maksimal dalam hitungan detik untuk mencapai tujuan. *waitTime* yang digunakan yaitu 0s untuk waktu minimal dan 120s untuk waktu tunggu maksimal yang berlaku untuk setiap *node.*

Group.waitTime = 0, 120

15. *Group.nrofInterface* mendefinisikan jumlah *interface* yang digunakan pada proses simulasi. Seperti pada penelitian ini yang menggunakan 1 *nterfaces* yaitu *wifi802.11b. Interface* yang digunakan menjadi hal penting dalam melakukan simulasi ini. Dimana *interfaces* digunakan sebagai media transmisi dalam pengiriman pesan. Pada penelitian ini akan menggunakan *interfaces* dengan tipe *wifi802.11b* karena, *wifi802.11b* memiliki cakupan area yang lebih luas dan kecepatan transmisi mencapai 11MB.

Group.nrofInterface = 1

16. *Group.speed* mendefinisikan kecepatan *node* yang digunakan dalam prose simulasi. Kecepatan *node* yang digunakan yaitu 20-40, 40-80 dan 80-160km/jam.

```
#konfigurasi kecepatan node yang digunakan
#2.7, 5.5   5.5, 11.1     11.1, 22.2
#20-40, 40-80, 80-160
Group.speed = 5.5, 11.1
```

17. *Group.nrofHosts* mendefinisikan jumlah *node* yang digunakan. Jumlah *node* yang digunakan disesuaikan dengan kebutuhan. Pada penelitian ini jumlah *node* yang digunakan yaitu 50, 100 dan 150 *node.* Jumlah ini dipilih berdasarkan jumlah maksimal pengunjung pada daerah simulasi Jalur Lintas Selatan Malang untuk 1 minggu

```
#konfigurasi jumlah node yang digunakan
Group.nrofHost = 150
```

18. *Group1.groupID* mendefinisikan nama dari *nrofHostGroup* yang digunakan yaitu p.

```
#group1 spesific setting
Group1.groupID = p
```

19. *Group1.routeFile* mendefinisikan *file* dengan ekstensi *.wkt* yang digunakan untuk pengujian dan letak *filenya* untuk *MapRouteMovement.*

```
Group1.routeFile = map/Peta.wkt
```

20. *Group1.routeType* mendefinisikan tipe *router* yang digunakan oleh *group1* yaitu *MapRouteMovement.*

```
Group1.routeType = 1
```

21. *Group1.waitTime* mendefinisikan waktu tunggu minimal dan maksimal dalam hitungan detik untuk mencapai tujuan. *waitTime* yang digunakan yaitu 0s untuk waktu minimal dan 120s untuk waktu tunggu maksimal yang berlaku untuk setiap *node* pada 1 *host.*

```
Group1.waitTime = 0, 120
```

22. *Events1.hosts* mendefinisikan *node* sumber dan *node* tujuan yaitu 0 sebagai *node* sumber dan 149 sebagai *node* tujuan.

```
#range of message source/destination addresses
Events1.hosts = 0,149
```

23. *Events1.prefix* mendefinisikan aliran dari pesan yang akan dikirimkan pada *group* yang sudah didefinisikan.

```
#message ID previx
Events1.previx = M
```

24. *MovementModel.rngSeed* mendefinisikan kecepatan awal yang digunakan untuk simulasi pada *MapRouteMovement.*

```
##movement model setting seed for movement model's
##pseudo random number generator (default 0)
MovemnetModels.rngSeed = 1
```

25. *MovementModel.worldSize* mendefinisikan ukuran peta yang digunakan untuk *mobility model* dalam penelitian ini. Ukuran peta yang digunakan berdasarkan ukuran pada peta asli yaitu 25Km atau 5000*5000m.

```
#World's size for Movement Models
##without implicit size (width, height; meters)
MovementModel.worldSize = 5000, 5000
```

26. *MovementModel.warmup* mendefinisikan waktu berapa lama untuk *host* berpindah dari peta pada daerah simulasi.

```
#How long time to move hosts in the world
##before real simulation
MovementModel.warmup = 1000
```

27. *MapBasedMovement.nrofMapFiles* mendefinisikan peta yang digunakan untuk simulasi.

```
##Map based movement – movement model
##specific settings
MapBasedMovement.nrofMapFiles = 1
```

28. *MapBasedMovement.mapFile1* mendefinisikan nama peta yang digunakan dengan *ekstensi file .wkt.*

```
MapBasedMovement.mapFile = map/Peta.wkt
```

29. *Report.nrofReports* mendefinisikan output untuk setiap selesainya simulasi yang dilakukan 1 kali untuk setiap simulasi.

```
#how many report to load
Report.nrofReports = 1
```

30. *Report.warmup* mendefinisikan hasil permulaan pada pengujian yang dilakukan.

```
#length of the warm up period (simulated seconds)
Report.warmup = 0
```

31. *Report.reportDir* mendefinisikan hasil *output* yang diperoleh dari simulasi yang berada pada *file reports/* pada *file ONE Simulator.*

```
#default directory of reports (can be overriden per Report with output
##setting)
Report.reportDir = reports/
```

32. *Report.report1* mendefinisikan nama untuk setiap simulasi dengan nama *MessageStatsReport.*

```
#report classes to load
Report.report1 = MessageStatsReport
```

33. *GUI.UnderlayImage.fileName* mendefinisikan alur *file* gambar yang diletakkan dibelakang proses simulasi.

```
#GUI underlay image settings
GUI.UnderlayImage.fileName = data/helsinki_underlay.png
```

34. *GUI.UnderlayImage.offset* mendefinisikan parameter untuk mengatur gambar latar pada proses simulasi.

```
#image offset in pixels (x, y)
GUI.UnderlayImage.offset = 64, 20
```

35. *GUI.UnderlayImage.scale* mendefinisikan parameter untuk mengetur skala latar belakang dari gambar.

```
#Scaling factor for the image
GUI.UnderlayImage.scale = 4.75
```

36. *GUI.UnderlayImage.rotate* mendefinisikan parameter untuk mengatur rotasi gambar dari latar belakang.

```
#image rotation (radians)
GUI.UnderlayImage.rotate = -0.015
```

37. *GUI.EventLogPanel.nrofEvents* mendefinisikan berapa banyak *event* yang akan ditampilkan pada *panel log.* Pada penelitian ini di buat *default* yaitu 30. *GUI.EventLogPanel.Refilter* mendefinisikan pola kelas dari *Java API* yang gunakan untuk *RE-matching.*

```
#how many events to show in the log panel (default = 30)
GUI.EventLogPanel.nrofEvents
#regular expression log filter (see pattern-class from the Java API for Re-matching details)
GUI.EventLogPanel.Refilter = .*p[1-9]<->p[1-9]$
```

38. *Events.nrof* mendefinisikan berapa banyak *event generator* yang digunakan dalam simulasi.

```
##message creation parameters
#how many event generators
Events.nrof = 1
```

39. *Events1.class* mendefinisikan tipe klas untuk *event generator.*

```
#class of the first event generator
Events1.class = MessageEventGenerator
```

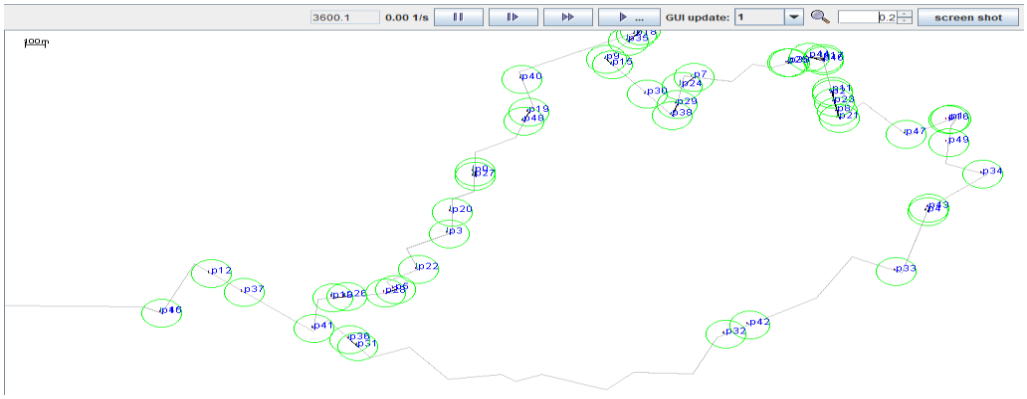
40. *Events1.size* mendefinisikan ukuran pesan yang digunakan dalam simulasi. Ukuran pesan ini adalah *range* maksimum yang digunakan untuk penelitian ini.

```
#konfigurasi ukuran pesan yang akan digunakan untuk dikirim
#Message sizes (IMB)
Events1.size = 1MB
```

41. *Events1.interval* mendefinisikan *interval* untuk *MessageEventGenerator* yang digunakan dan didefinisikan dalam detik antara 25-35 detik.

```
#(following settings are specific for the MessageEventGenerator class)
#creation interval in seconds (one new message every 25 to 35 seconds)
Events1.interval = 25, 35
```

Setelah semua konfigurasi sudah dilakukan, kita bisa menjalankan *ONE Simulator* dengan mengetikkan perintah *one.bat* pada *command prompt* dan akan muncul tampilan seperti dibawah ini:

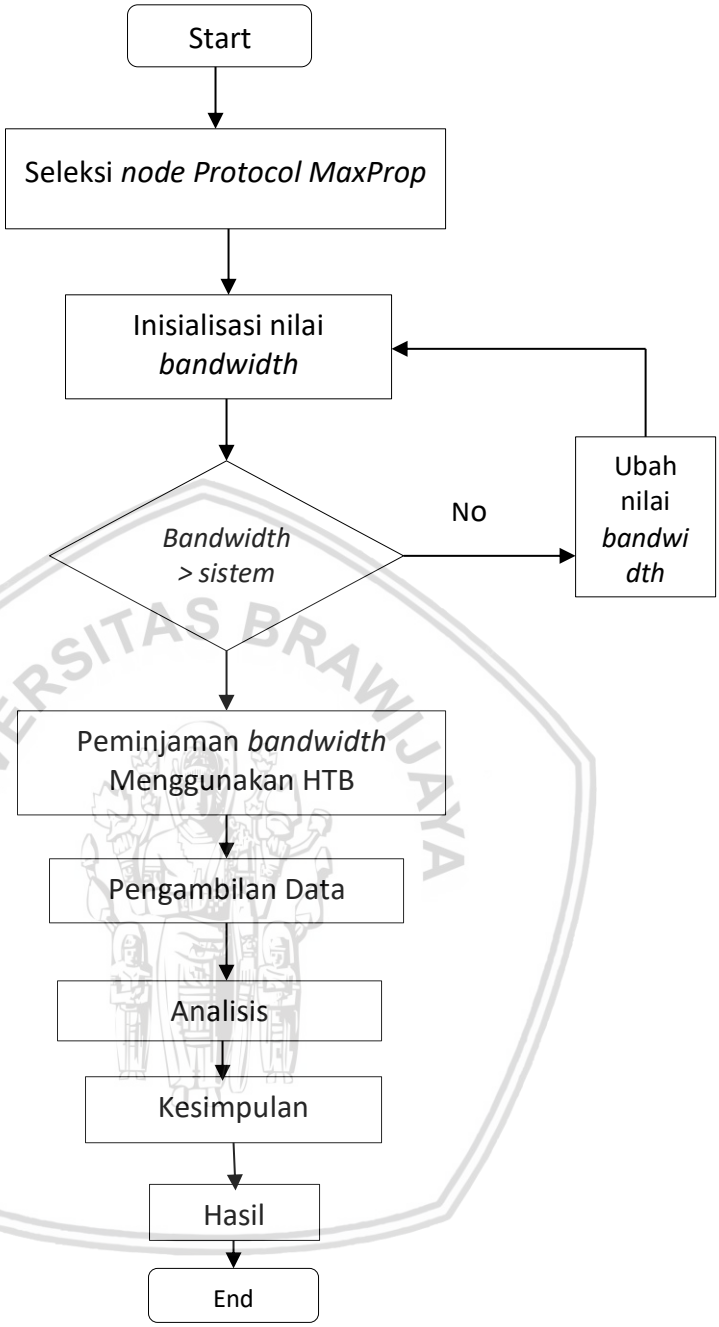


Gambar 5. 3 Proses Pengujian Untuk *Protocol Maxprop* Dengan 50 *Node*

Gambar 5.3 merupakan hasil pengujian *protocol Maxprop* dengan jumlah *node* 50 dan kecepatan *node* 80-160 km/jam.

5.1.1.2 Skenario *Algoritme Hierarchical Token Bucket* Untuk Seleksi *Node Routing Multi copy*

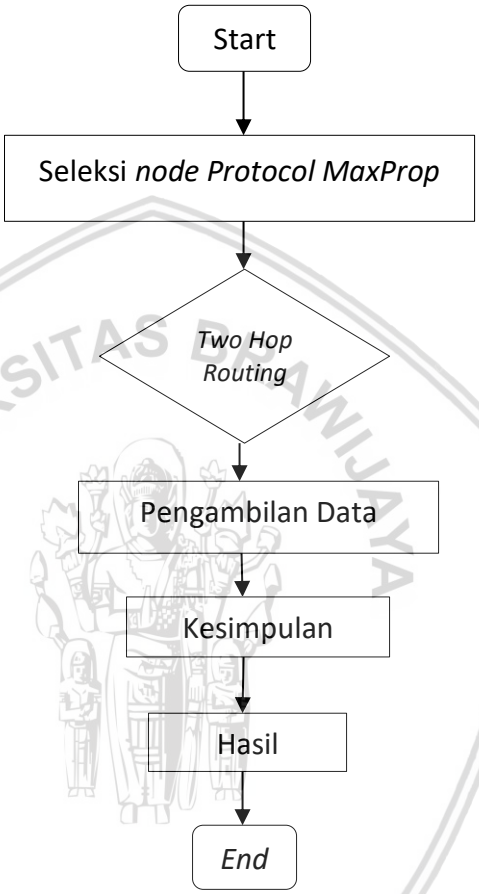
Pada tahap ini terdapat perancangan untuk penerapan *Algoritme Hierarchical Token Bucket* yang akan digunakan untuk seleksi *node routing multi copy*. Gambar 5.4 menunjukan diagram alir *Algoritme Hierarchical Token Bucket,* yaitu:



Gambar 5. 4 Diagram Alir *Algoritme Hierarchical Token Bucket*

Gambar 5.4 menunjukan cara kerja dari alokasi *bandwidth* dengan menggunakan *algoritme Hierarchical Token Bucket* dengan melakukan seleksi *node* terlebih dahulu pada *protocol MaxProp*. Sebelum melakukan peminjaman *bandwidth,* dilakukan inisialisasi *bandwidth* terlebih dahulu, hal ini ditujukan untuk membatasi jumlah *bandwidth* yang diminta agar tidak melebihi dari kapasitas sistem. Ketika jumlah *bandwidth* yang diminta melebihi kapasitas sistem, maka dilakukan perubahan pada jumlah *bandwidth* yang diminta. Setelah jumlah *bandwidth* yang diminta lebih kecil

dari nilai *bandwidth* sistem maka, peminjaman *bandwidth* dengan menggunakan *Algoritme Hierarchical Token Bucket* akan dilakukan. Proses berikutnya yaitu pengambilan data dari hasil peminjaman *bandwidth* menggunakan *Algoritme Hieararcial Token Bucket.* hasil ini akan digunakan untuk proses analisis untuk menentukan kesimpulan pada pengujian dan akan didapatkan hasil dari pengujian yang dilakukan. Gambar 5.5 menunjukan proses dari seleksi *node* pada *routing multi copy* yaitu:



Gambar 5. 5 Diagram Alir Seleksi *Node Routing Multi copy*

Proses seleksi *node routing multi copy* dilakukan menggunakan mekanisme *two hop routing* pada *Delay Tolerant Network* yaitu pesan hanya akan dikirimkan pada *node* yang menjadi tujuan dari *node* sumber. Proses berikutnya yaitu pengambilan data dari hasil peminjaman *bandwidth* menggunakan *Algoritme Hieararcial Token Bucket.* hasil ini digunakan untuk proses analisis untuk menentukan kesimpulan pada pengujian. Pada seleksi *node* ini akan dibagi berdasarkan *node* ganjil dan genap yaitu pesan akan dikirimkan pada *node* genap dan ganjil. Hal ini dilakukan karena mekanisme *two hop routing* yang menerapkan pengiriman pesan berdasarkan 2 jalur yang berbeda dengan tujuan yang sama.

Berikut adalah proses seleksi *node* untuk *protocol MaxProp* dengan *two hop routing* yaitu:

- Seleksi *node* akan dilakukan untuk 50, 100, 150 dan 200 *node*.
- *Node* sumber yaitu index 0 dan *node* tujuannya adalah jumlah batas *node* yang sudah ditentukan.
- Seleksi *node* menggunakan mekanisme *two hop routing* untuk *protocol MaxProp*.
- Seleksi *node* dilakukan dengan pemilihan jalur ganjil dan genap untuk jumlah *node* yang ditentukan, seperti pada tabel 5.2 untuk *node* ganjil dan *node* genap, tabel 5.3 untuk *node* ganjil dan *node* genap, tabel 5.4 untuk *node* ganjil dan *node* genap dan tabel 5.5 untuk *node* ganjil dan *node* genap, yaitu:

Tabel 5. 2 Seleksi Mekanisme *Two Hop Routing* 50 *Node*

Jumlah node	Jalur yang dilalui
50	0-2-4-6-8-10-12-14-16-18-20-22-24-26-28-30-32-34-36-38-40-42-44-46-48
	0-1-3-5-7-9-11-13-15-17-19-21-23-25-27-29-31-33-35-37-39-41-43-45-47-49

Tabel 5. 3 Seleksi Mekanisme *Two Hop Routing* 100 *Node*

Jumlah node	Jalur yang dilalui
100	0-2-4-6-8-10-12-14-16-18-20-22-24-26-28-30-32-34-36-38-40-42-44-46-48-50-52-54-56-58-60-62-64-66-68-70-72-74-76-78-80-82-84-86-88-90-92-94-96-98
	0-1-3-5-7-9-11-13-15-17-19-21-23-25-27-29-31-33-35-37-39-41-43-45-47-49-51-53-55-57-59-61-63-65-67-69-71-73-75-77-79-81-83-85-87-89-91-93-95-97-99

Tabel 5. 4 Seleksi Mekanisme *Two Hop Routing* 150 *Node*

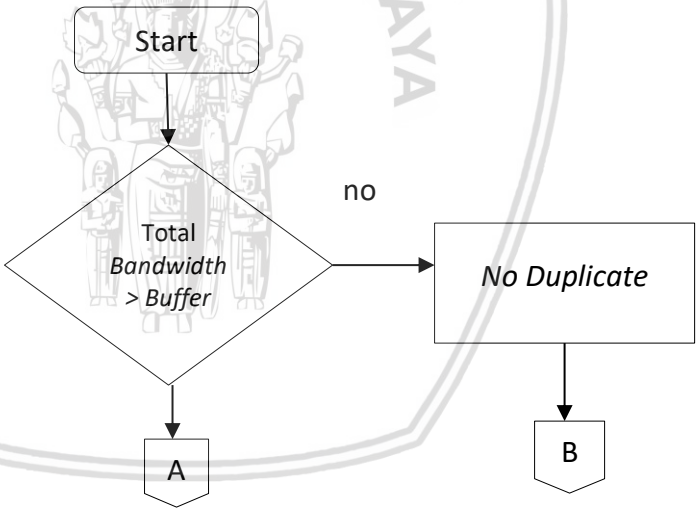
Jumlah node	Jalur yang dilalui
150	0-2-4-6-8-10-12-14-16-18-20-22-24-26-28-30-32-34-36-38-40-42-44-46-48-50-52-54-56-58-60-62-64-66-68-70-72-74-76-78-80-82-84-86-88-90-92-94-96-98-100-102-104-106-108-110-112-114-116-118-120-122-124-126-128-130-132-134-136-138-140-142-144-146-148
	0-1-3-5-7-9-11-13-15-17-19-21-23-25-27-29-31-33-35-37-39-41-43-45-47-49-51-53-55-57-59-61-63-65-67-69-71-73-75-77-79-81-83-85-87-89-91-93-95-97-99-101-103-105-107-109-111-113-115-117-119-121-123-125-127-129-131-133-135-137-139-141-143-145-147-149

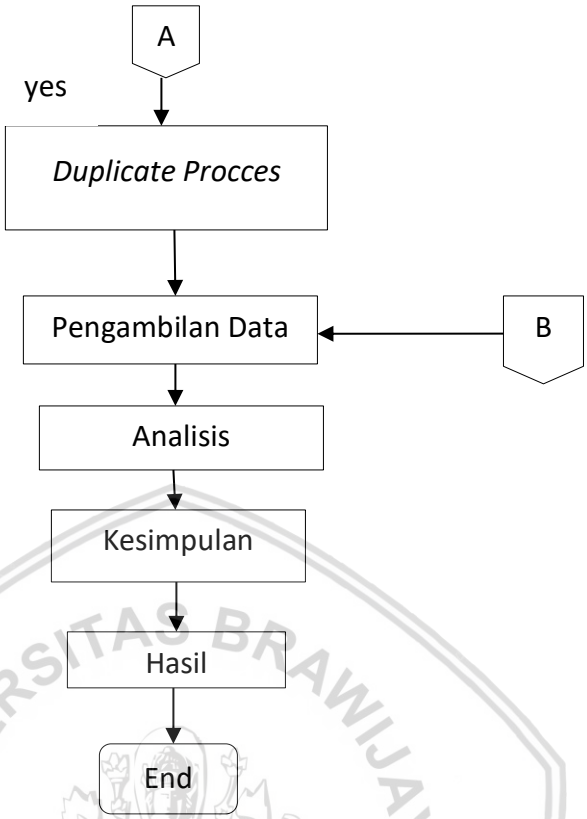
Tabel 5. 5 Seleksi Mekanisme *Two Hop Routing 200 Node*

Jumlah node	Jalur yang dilalui
200	0-2-4-6-8-10-12-14-16-18-20-22-24-26-28-30-32-34-36-

	38-40-42-44-46-48-50-52-54-56-58-60-62-64-66-68-70-72-74-76-78-80-82-84-86-88-90-92-94-96-98-100-102-104-106-108-110-112-114-116-118-120-122-124-126-128-130-132-134-136-138-140-142-144-146-148-150-152-154-156-158-160-162-164-166-168-170-172-174-176-178-180-182-184-186-188-190-192-194-196-198
	0-1-3-5-7-9-11-13-15-17-19-21-23-25-27-29-31-33-35-37-39-41-43-45-47-49-51-53-55-57-59-61-63-65-67-69-71-73-75-77-79-81-83-85-87-89-91-93-95-97-99-101-103-105-107-109-111-113-115-117-119-121-123-125-127-129-131-133-135-137-139-141-143-145-147-149-151-153-155-157-159-161-163-165-167-169-171-173-175-177-179-181-183-185-187-189-191-193-195-197-199

Hasil dari seleksi *node* digunakan untuk melakukan peminjaman *bandwidth* dengan menggunakan *Algoritme Hierarchical Token Bucket*. Dari proses peminjaman *bandwidth* akan dilakukan manajemen *bandwidth.* Gambar 5.6 menjelaskan alur dari proses *duplicate* pada pengujian yang akan dilakukan, yaitu:





Gambar 5. 6 *Flowchart Duplicate Prosses*

Ketika proses *duplicate* dilakukan, maka proses peminjaman *bandwidth* akan berhenti pada *index* tertentu karena total *bandwidth* yang diperoleh melebihi dari *buffer.* Hal ini akan berakibat pada beberapa *node* yang tidak menerima sejumlah *bandwidth* yang disediakan. *Node* yang bisa melakukan peminjaman *bandwidth* secara tidak langsung akan memperoleh pesan yang lebih banyak dibandingkan dengan *node* yang tidak bisa melakukan peminjaman *bandwidth.* Namun ketika proses *duplicate* tidak terjadi maka proses peminjaman *bandwidth* akan terus dilakukan sampai jumlah *index node* tertentu. Ketika proses *duplicate* dilakukan maka, *duplicate* dari *bandwidth* yang diperoleh untuk setiap *node* nya akan dihapus dan ketika proses *duplicate* tidak terjadi, maka terdapat pemberitahuan pada sisi *server* bahwa *duplicate* dari *bandwidth* berhasil terkirim.

Pada tabel 5.6 mendeskripsikan tentang cara kerja dari *Algoritme Hierarchical Token Bucket* untuk peminjaman *bandwidth. Bandwidth* yang dipinjam tidak boleh melebihi kapasitas *sistem.* Berikut adalah *source code* untuk kondisi *Algoritme Hierarchical Token Bucket.*

Tabel 5. 6 *Source Code Algoritme Hierarchical Token Bucket*

```
1  if                    (!isInputAction)                    {
2  System.out.println("Peminjaman Node GENAP :");
3                     System.out.print("Masukan Jumlah Bandwidth
4  :");
5  inputSave = masukan.nextInt();
6                                  bandwidthpinjam           =
   inputSave;
```

```
7                                       isInputAction = true
                                    } else {//kondisi true
8                                       bandwidthpinjam           =
9   inputSave;
                                    }
10              if (flag) {
11                              flag = false;
                            }
12                      System.out.println("pinjam       "       +
13  bandwidthpinjam + " Mbps ");
14                      if (bandwidthpinjam > sistem) {
15                          System.out.println("Permintaan   Tidak
16  Dilayani");
                        }
17                      if (bandwidthpinjam < sistem) {
18                          System.out.println("Peminjaman
    Kelas");
19                          if     ((ceil     ==     rate))     {
20  System.out.println("Permintaan Tidak Dilayani");
21                          }              else               {
    System.out.println("Permintaan  Akan Diproses");
22                          }
                        }
23                      if ((bandwidthpinjam > rate)) {
24                          System.out.println("Pesan di Drop");
```

Penjelasan:

1. Baris 1 yaitu perulangan *if,* dimana *method* isInputAction bernilai *else.* Hal ini dilakukan untuk mempermudah memanggil *variable* selanjutnya supaya bernilai sama dengan apa yang di *input client.*
2. Baris 2-4 perintah untuk mencetak Peminjaman *Node* Genap dan masukan Jumlah *Bandwidth*.
3. Baris 5 yaitu *variable* yang digunakan untuk menyimpan *input* data yang dimasukkan oleh *client* dengan menggunakan tipe data *integer.*
4. Baris 6-7 yaitu mendeklarasikan nilai dari bandwidthpinjam = inputSave.
5. Baris 8yaitu kondisi *true* untuk isInputAction dan perulangan *else.*
6. Baris 9-12 mendeklarasikan nilai dari bandwidthpinjam = inputSave.
7. Baris 13-14 yaitu kondisi dimana jumlah *bandwidth* yang diminta akan di *convert* langsung menjadi Mbps. Dan perintah untuk mencetak *bandwidth* yang dipinjam.
8. Baris 15-16 kondisi *if,* dimana bandwidthpinjam>sistem dan perintah untuk mencetak Permintaan Tidak Dilayani.
9. Baris 17-19 yaitu kondisi *if,* dimana bandwidthpinjam < sistem dan perintah untuk mencetak Peminjaman Kelas.
10. Baris 20-21 yaitu perulangan *if,* dimana nilai ceil == rate dan perintah untuk mencetak Permintaan Tidak Dilayani.
11. Baris 22-23 yaitu kondisi *else,* dan perintah untuk mencetak Permintaan Akan Diperoses.
12. Baris 24-26 yaitu perulangan *if,* dimana nilai bandwidthpinjam > rate dan perintah untuk mencetak Pesan Di *Drop*.

Pada *implementasi Algoritme Hierarchical Token Bucket* terdapat dua proses peminjaman kelas yaitu peminjaman *bandwidth* pertama dan yang kedua. Hal ini ditujukan untuk mendapatkan nilai *bandwidth* maksimal untuk setiap *node*nya. Tabel 5.7 adalah *source code* untuk proses peminjaman

bandwidth yang pertama menggunakan *Algoritme Hierarchical Token Bucket.*

Tabel 5. 7 *Source Code* Peminjaman *Bandwidth* Pertama *Node* Ganjil Dan *Node* Genap

```
1                totalnode1      =      bandwidthasal      +
     bandwidthpinjam;
2                System.out.println("Total Node " + node1gjl
     + " Mbps = " + bandwidthasal + " Mbps + " + bandwidthpinjam
     + " Mbps = " + totalnode1 +" Mbps");
3                totalnode2      =      bandwidthasal      -
     bandwidthpinjam;
4                System.out.println("Total Node " + node2gjl
     + " Mbps = " + bandwidthasal + " Mbps - " + bandwidthpinjam
     + " Mbps = " + totalnode2 +" Mbps");
5                totalnode3 = bandwidthasal + sisabandwidth;
                 System.out.println("Total Node " + node3gjl
     + " Mbps = " + bandwidthasal + " Mbps + " + sisabandwidth +
     " Mbps = " + totalnode3 +" Mbps");            //PEMINJAMAN
6    KELAS PERTAMA NODE GENAP
7                totalnode1      =      bandwidthasal      +
     bandwidthpinjam;
8                System.out.println("Total Node " + node1gnp
     + " = " + bandwidthasal + " Mbps + " + bandwidthpinjam + "
     Mbps = " + totalnode1 + " Mbps");
9                totalnode2      =      bandwidthasal      -
     bandwidthpinjam;
10               System.out.println("Total Node " + node2gnp
     + " = " + bandwidthasal + " Mbps - " + bandwidthpinjam + "
     Mbps = " + totalnode2 + " Mbps");
11               totalnode3 = bandwidthasal + sisabandwidth;
12               System.out.println("Total Node " + node3gnp
     + " = " + bandwidthasal + " Mbps + " + sisabandwidth + "
     Mbps = " + totalnode3 + " Mbps");
```

Penjelasan:

1. Baris 1-3 perintah untuk mencetak totalnode1 untuk *node* ganjil yaitu bandwidthasal + bandwidthpinjam.
2. Baris 4-6 perintah untuk mencetak totalnode2 untuk *node* ganjil yaitu bandwidthasal– bandwidthpinjam.
3. Baris 7 perintah untuk mencetak totalnode3 untuk *node* ganjil yaitu bandwidthasal + sisabandwidth.
4. Baris 8-9 perintah untuk mencetak totalnode1 untuk *node* genap yaitu bandwidthasal + bandwidthpinjam.
5. Baris 10-11 perintah untuk mencetak totalnode2 untuk *node* genap yatu bandwidthasal – bandwidthpinjam.
6. Baris 12 perintah untuk mencetak totalnode3 untuk *node* genap yaitu bandwidthasal + sisabandwidth.

Tabel 5.8 adalah *source code* untuk peminjaman *bandwidth* kedua *node* ganjil dan genap pada *Algoritme Hierarchical Token Bucket,* yaitu:

Tabel 5. 8 *Source Code* Peminjaman *Bandwidth* Kedua *Node* Ganjil Dan Genap

```
1                totalnode1 = totalnode2 + bandwidthpinjam +
     bandwidthasal;
2                System.out.println("Total Node" + node1gnp +
     " Mbps = " + totalnode2 + " Mbps + " + bandwidthpinjam + "
     Mbps + " + bandwidthasal + " Mbps = " + totalnode1 + "
```

```
3       Mbps");
                totalnode2 = totalbandwidth - bandwidthpinjam
4       + bandwidthasal;
                System.out.println("Total Node" + node2gnp +
        " Mbps = " + totalbandwidth + " Mbps + " + bandwidthasal + "
5       Mbps - " + bandwidthpinjam + " Mbps = " + totalnode2 + "
6       Mbps");
                totalnode3 = bandwidthasal + bandwidthpinjam;
                System.out.println("Total Node" + node3gnp +
        " Mbps = " + bandwidthasal + " Mbps + " + bandwidthpinjam + "
7       Mbps = " + totalnode3 + " Mbps");
        //PEMINJAMAN KELAS KEDUA NODE GENAP
8               totalnode1 = totalnode2 + bandwidthpinjam +
        bandwidthasal;
                System.out.println("Total Node" + node1gnp + " =
9       " + totalnode2 + " + " + bandwidthpinjam + " + " +
        bandwidthasal + " = " + totalnode1);
10              totalnode2 = totalbandwidth - bandwidthpinjam +
        bandwidthasal;
                System.out.println("Total Node" + node2gnp + " =
11      " + totalbandwidth + " + " + bandwidthasal + " - " +
12      bandwidthpinjam + " = " + totalnode2);
                totalnode3 = bandwidthasal + bandwidthpinjam;
                System.out.println("Total Node" + node3gnp + " =
        " + bandwidthasal + " + " + sisabandwidth + " = " +
        totalnode3);
```

Penjelasan:

- Bandwidthasal yaitu jumlah asal *bandwidth* untuk setiap *node* yaitu 1MB.
- Bandwidthpinjam yaitu jumlah *bandwidth* yang dipinjam oleh *node.*
- Sisabandwidth yaitu diperoleh dari sistem – bandwidthpinjam. Dimana sisabandwidth ini akan didistribusikan langsung pada *node* ke 3 dalam proses peminjaman.
- Totalbandwidth yaitu diperoleh dari bandwidthasal + sisabandwidth yang kemudian akan digunakan untuk *node* ke 2 pada proses peminjaman yang kedua kalinya.
- Bandwidthmaksimal yaitu jumlah maksimal *bandwidth* yang bisa dipinjam.

1. Baris 1-3 perintah untuk mencetak totalnode1 untuk *node* ganjil pada peminjaman kelas yang kedua yaitu totalnode2 + bandwidthpinjam + bandwidthasal.
2. Baris 4-5 perintah untuk mencetak totalnode2 untuk *node* ganjil pada peminjaman kelas yang kedua yaitu totalbandwidth – bandwidthpinjam + bandwidthasal.
3. Baris 6-7 perintah untuk mencetak totalnode3 untuk *node* ganjil pada peminjaman kelas yang kedua yaitu bandwidthasal + bandwidthpinjam.
4. Baris 8-10 perintah untuk mencetak totalnode1 untuk *node* genap pada peminjaman kelas yang kedua yaitu totalnode2 + bandwidthpinjam + bandwidthasal.
5. Baris 11-12 perintah untuk mencetak totalnode2 untuk *node* genap pada peminjaman kelas yang kedua yaitu totalbandwidth – bandwidthpinjam + bandwidthasal.
6. Baris 13-14 perintah untuk mencetak totalnode3 untuk *node* genap pada peminjaman kelas yang kedua yaitu bandwidthasal+ bandwidthpinjam.

Pada *implementasi Algoritme Hierarchical Token Bucket* pembuatan *node* dilakukan menggunakan struktur data *tree.* Struktur data *tree* yang digunakan yaitu *AVL Tree*. Tabel 5.9 adalah *source code* untuk pembuatan *node* pada *Algoritme Hierarchical Token Bucket.*

Tabel 5. 9 *Source Code* AVL Tree

```
1   avlNode avl = new avlNode();
2           //insert data Tree menggunakan fungsi bantuan
3           int awal = 0;
4           for (int i = awal; i < 50; i++) {
5               avl.insert(i);
6           }
7           System.out.println("Final    Self    balanced    AVL
    Tree:\n");
8           avl.preOrderDisplay();
9           System.out.println("");
```

Penjelasan:

1. Baris 1-2 yaitu deklarasi *method* dengan nama avlNode().
2. Baris 3 yaitu deklarasi *method* dengan nama awal menggunakan tipe data *integer.*
3. Baris 4 yaitu perulangan *for* dengan tipe data *integer* yaitu i = awal dan i < 50.
4. Baris 5-6 yaitu deklarasi *method* dengan nama *avl.insert(i)*.
5. Baris 7 yaitu perintah untuk mencetak Final Self Balanced AVL Tree.
6. Baris 8-9 yaitu deklrasi *method* dengan nama *avl.preOrderDisplay();*

Pada penelitian ini dilakukan seleksi *node* dengan menggunakan mekanisme *two hop routing* yaitu pesan hanya akan dikirimkan pada *node* yang menjadi tujuan dari *node* sumber. Seleksi *node* berdasarkan *index node* ganjil dan genap. Dimana proses peminjaman ditentukan berdasarkan *node* yang terdekat dengan *node* sumber. Jika *node* terdekat *node* sumber angka ganjil, maka proses peminjaman *bandwidth* akan dilakukan untuk *node* ganjil terlebih dahulu kemudian dilanjutkan dengan *node* genap. Tabel 5.10 adalah *source code* untuk seleksi *node* ganjil dan *node* genap yaitu:

Tabel 5. 10 *Source Code* Seleksi *Node*

```
1   System.out.print("Node Ganjil :  ");
2           for (int j = 1; j < 50; j++) {
3               {
4                   System.out.print(j + " ");
5   j = j + 1;}}
6   System.out.println("");
7            System.out.print("Node Genap  :  ");
8            for (int j = 0; j < 50; j++) {
9                {
10                   System.out.print(j + " ");
11                   j = j + 1;
12               }
13           }
14           System.out.println("");
     System.out.println("============================");
15   //PENGIRIMAN DARI NODE SUMBER KE NODE TUJUAN
16           if (source == 0) {
17               System.out.print("Node  terdekat  yaitu  =
```

```
18      ");
19              }
20              System.out.println(awal + 1 + "ini datanya");
                if ((awal + 1) % 2 == 0) {
                 System.out.println("genap");
```

Penjelasan:

1. Baris 1 yaitu perintah untuk mencetak Node Ganjil.
2. Baris 2-3 yaitu perulangan *for* menggunakan tipe data *integer,* dimana j =1 dan j < 50.
3. Baris 4-5 yaitu perintah untuk mencetak nilai j dimana nilai j diperoleh dari j+1.
4. Baris 6 yaitu perintah untuk mencetak Node Genap.
5. Baris 7-8 yaitu perulangan *for* menggunakan tipe data *integer,* dimana j =1 dan j < 50.
6. Baris 9-13 perintah untuk mencetak nilai j dimana nilai j diperoleh dari j+1.
7. Baris 14-15 perulangan *if,* dimana nilai source == 0.
8. Baris 16-17 yaitu perintah untuk mencetak Node Terdekat Yaitu
9. Baris 18 yaitu perintah untuk mencetak nilai awal + 1 untu nilai datanya.
10. Baris 19yaitu perulangan *if,* dimana nilai awal + 1 mod 2 ==0, maka
11. Baris 20 perintah untuk mencetak Genap.

Ketika proses peminjaman *bandwidth* sudah selesai, maka jumlah *bandwidth* yang sudah dipinjam akan dihitung untuk jumlah keseluruhan *node*. Proses ini akan dilakukan setiap kali proses peminjaman *bandwidth* untuk *node* ganjil atau *node* genap. Tabel 5.11 adalah *source code* untuk perhitungan *bandwidth* pada *node* ganjil dan *node* genap yaitu:

Tabel 5. 11 *Source Code* Perhitungan *Bandwidth* Untuk *Node* Ganjil Dan Genap

```
1   //DEKLARASI INDEX NODE GANJIL
2               isibandwidth[node1gjl-1] = totalnode1;
3               isibandwidth[node2gjl-1] = totalnode2;
4               isibandwidth[node3gjl-1] = totalnode3;
5               //PERHITUNGAN HASIL BANDWIDTH UNTUK NODE GANJIL
                System.out.println("Total  Bandwidth  Untuk  Setiap
6   Node :");
                if (totalnode1 > 1024) {
7                   System.out.println("Bandwidth    Node    "    +
8   node1gjl + " = " + totalnode1 + "  ");
                }
9               if (totalnode2 > 1024) {
10                  System.out.println("Bandwidth    Node    "    +
11  node2gjl + " = " + totalnode2 + " ");
                }
12              if (totalnode3 > 1024) {
13                  System.out.println("Bandwidth    Node    "    +
14  node3gjl + " = " + totalnode3 + " ");
                }
15              //PERTUKARAN POSISI UNTUK NODE GANJIL
16              idxgjl = 1;
17              if (idxgjl == 1) {
18                  node1gjl = node2gjl;
19                  node2gjl = node3gjl;
20                  node3gjl = node3gjl - 2;
```

```
21              }
          }  while  (node3gjl!=-1);//PERULANGAN  AKAN  BERHENTI
22  KETIKA INDEX NODE BERADA PADA NILAI 1
           //DEKLARSI INDEX UNTUK NODE GENAP
23              isibandwidth[node1gnp-1] = totalnode1;
24              isibandwidth[node2gnp-1] = totalnode2;
25              isibandwidth[node3gnp-1] = totalnode3;
26              //PERHITUNGAN HASIL BANDWIDTH UNTUK NODE GENAP
27              System.out.println("Total  Bandwidth  Untuk  Setiap
    Node :");
28              if (totalnode1 > 1024) {
                    System.out.println("Bandwidth    Node    "    +
29  node1gnp + " = " + totalnode1 + "  ");
30              }
                if (totalnode2 > 1024) {
31                  System.out.println("Bandwidth    Node    "    +
32  node2gnp + " = " + totalnode2 + " ");
33              }
                if (totalnode3 > 1024) {
34                  System.out.println("Bandwidth    Node    "    +
35  node3gnp + " = " + totalnode3 + " ");
36              }
    total += totalnode1 + totalnode2 + totalnode3;
37                      System.out.println("Total Mbps " + total);
38                      if (total > buffer) {
39                          break;
                        }
40              //PERTUKARAN POSISI UNTUK NODE GENAP
41              idxgnp = 1;
42              if (idxgnp == 1) {
43                  node1gnp = node2gnp;
                    node2gnp = node3gnp;
44                  node3gnp = node3gnp - 2;
45              }
46          }  while  (node3gnp!=0);//PERULANGAN  AKAN  BERHENTI
47  KETIKA INDEX PADA NODE GENAP PADA NILAI 2
      }
48
```

Penjelasan:

1. Baris 1-2 deklarsi dari isibandwidth untuk node1gjl-1 yaitu sama dengan totalnode1.
2. Baris 3 deklarsi dari isibandwidth untuk node1gjl-1 yaitu sama dengan totalnode2.
3. Baris 4 deklarsi dari isibandwidth untuk node1gjl-1 yaitu sama dengan totalnode3.
4. Baris 5-6 perintah untuk mencetak Total *bandwidth* untuk setiap *node* ganjil.
5. Baris 7-9 perulangan *if,* dimana nilai untuk totalnode1>1024 untuk proses peminjaman *bandwidth* yang pertama dan kedua dan perintah untuk mencetak *bandwidth node* untuk node1gjl.
6. Baris 10-12 perulangan *if* yaitu nilai untuk totalnode2>1024 untuk proses peminjaman *bandwidth* yang pertama dan kedua dan perintah untuk mencetak *bandwidth node* untuk node2gjl.
7. Baris 13-15 perulangan *if* yaitu nilai untuk totalnode3>1024 untuk proses peminjaman *bandwidth* yang pertama dan kedua dan perintah untuk mencetak *bandwidth node* untuk node3gjl.
8. Baris 16-17 yaitu deklarasi *index* untuk *node* ganjil dengan nilai 1.

9. Baris 18 yaitu perulangan *if,* dimana idxgjl == 1.
10. Baris 19-21 pertukaran posisi untuk node1gjl = node2gjl, node2gjl = node3gjl dan node3gjl = node3gjl - 2.
11. Baris 22-23 perulangan *while,* dimana idxgjl akan berhenti ketika berada pada index -1.
12. Baris 24 deklarsi dari isibandwidth untuk node1gnp-1 yaitu sama dengan totalnode1.
13. Baris 25 deklarsi dari isibandwidth untuk node2gnp-1 yaitu sama dengan totalnode2.
14. Baris 26 deklarsi dari isibandwidth untuk node2gnp-*1* yaitu sama dengan totalnode3.
15. Baris 27-28 perintah untuk mencetak total *bandwidth* untuk setiap *node* genap.
16. Baris 29-31 perulangan *if* yaitu nilai untuk totalnode1>1024 untuk proses peminjaman *bandwidth* yang pertama dan kedua dan perintah untuk mencetak *bandwidth node* untuk node1gnp.
17. Baris 32-34 perulangan *if* yaitu nilai untuk totalnode2>1024 untuk proses peminjaman *bandwidth* yang pertama dan kedua dan perintah untuk mencetak *bandwidth node* untuk node2gnp.
18. Baris 35-37 perulangan *if* yaitu nilai untuk totalnode3>1024 untuk proses peminjaman *bandwidth* yang pertama dan kedua dan perintah untuk mencetak *bandwidth node* untuk node3gnp.
19. Baris 38 yaitu proses penjumlahan untuk hasil *bandwidth* yang diperoleh oleh totalnode1, totalnode2 dan totalnode3.
20. Baris 39 perintah untuk mencetak Total Mbps yang diperoleh oleh setiap *node.*
21. Baris 40-42 yaitu perulangan *if,* dimana jika nilai dari total > buffer maka proses peminjaman *bandwidth* akan berhenti.
22. Baris 43 yaitu deklarasi index untuk *node* genap dengan nilai 1.
23. Baris 44 yaitu perulangan *if,* dimana idxgnp == 1.
24. Baris 45-47 pertukaran posisi untuk node1gnp = node2gnp, node2gnp = node3gnp dan node3gnp = node3gnp - 2.
25. Baris 48-49 perulangan *while,* dimana idxgnp akan berhenti ketika berada pada index 0.

5.1.2 Perancangan Sistem

Pada tahap ini akan dilakukan perancangan untuk sistem pada *Algoritme Hierarchical Token Bucket*. Sistem disini terdiri beberapa *subsistem* seperti *subsistem server, subsistem intermediatte node* dan *subsistem client. Subsistem server* berperan sebagai penyedia layanan peminjaman *bandwidth, subsitem client* berperan sebagai *client* atau pengguna dari layanan dan *subsistem intermediatte node* berperan sebagai penghubung antara *subsistem server* dan *subsistem client* dalam pengiriman pesan. Berikut adalah *source code* dari *bundle layer* yang digunakan pada penelitian ini, yaitu:

1. *Delay Tolerant Network Server*

Pada tahap ini menjelaskan tentang alur pengiriman pesan yang dilakukan oleh *server.*

Tabel 5.12 adalah *source code* dari *Delay Tolerant Network* pada sisi *server,* yaitu:

Tabel 5. 12 *Source Code Bundle Delay Tolerant Netwrok* Pada Sisi *Server*

```
1   public class Server {
2       private static ServerSocket server;
3   private static int port = 9876;
4       public static void main(String args[]) throws
    IOException, ClassNotFoundException {
5           System.out.println("SERVER");
6           server = new ServerSocket(port);
7           Scanner a = new Scanner(System.in);
8           while (true) {
9               System.out.println("Waiting for client
    request");
10              Socket socket = server.accept();
11              ObjectInputStream ois = new
    ObjectInputStream(socket.getInputStream());
12              System.out.print("Input : ");
13              System.out.println("Send Bundle ");
14            String message = (String) ois.readObject();
15              System.out.println("Message Received: " +
    message);
16              ObjectOutputStream oos = new
    ObjectOutputStream(socket.getOutputStream());
17              oos.writeObject("Hi Client " + message);
18              Client c = new Client();
19              if (!c.getFlag()) {
20                  System.out.println("PROCESS... " +
21  message);
22              }
23  System.out.println(buffer + " Mbps ");
24              if(c.getTotalBuffer() > 1024){
25              System.out.println("Proses Berhenti ");
26          }
27           System.out.println(c.getduplicateStatus());
28              if(totalBuffer > buffer){
29                          dp = "y";
30                      }
31                      else{
32                      dp = "n";
33                      }
34              if (c.getDp().equalsIgnoreCase("")) {
35                  System.out.println("No Duplicate");
36              } else {
37                  System.out.println("Duplicate");
38              }
39              ois.close();
40              oos.close();
41              socket.close();
42              if (message.equalsIgnoreCase("exit")) {
43                  break;
44              }
45          }
46          System.out.println("Closing connecteions");
            server.close();
```

Penjelasan:

1. Baris 1 yaitu deklarasi *class* dengan nama Server.
2. Baris 2 yaitu deklarasi *variable* ServerSocket berupa *static.*

3. Baris 3 yaitu deklarsi untuk *port* yang akan digunakan oleh *Server* yaitu *port* 9876 dengan tipe *int.*
4. Baris 4 yaitu deklarasi *main method* dari *class Server.*
5. Baris 5 perintah untuk mencetak *SERVER*.
6. Baris 6-7 yaitu deklarasi untuk membuat *socket* baru dengan nama ServerSocket beserta *port* yang digunakan untuk berkomunikasi dengan *client.*
7. Baris 8-9 yaitu perulangan *while,* dimana jika proses bernilai *true,* maka akan mencetak *Waiting for client request.*
8. Baris 10 yaitu deklarasi untuk membuat *socket* dan menunggu koneksi dari *client* menggunakan *method* server.accept().
9. Baris 11 yaitu membaca *socket* dari ObjectInputStream *method.*
10. Baris 12 yaitu perintah untuk mencetak *Input.*
11. Baris 13 perintah untuk mencetak *Send Bundle*.
12. Baris 14-15 yaitu deklarsi pesan yang akan dikirim oleh *server* bernilai *String* dan perintah untuk mencetak *Message Received* pada sisi *server.*
13. Baris 16 yaitu deklarasi *method* ObjectOutputStream.
14. Baris 17 yaitu deklarsi pesan yang akan dikirimkan dari *server* ke *client* berupa *Hi Client + message*.
15. baris 18-19 yaitu deklarasi *method* dengan nama *Client().*
16. Baris 20-21 yaitu perulangan *if,* dimana ketika 1 klai proses pada *server* akan mengambil nilai menggunakan *method* getFlag() dan perintah untuk mencetak *PROCESS ...+ message*.
17. Baris 22 digunakan untuk mencetak nilai buffer + Mbps.
18. Baris 23 yaitu perulangan *if,* dimana mengambil nilai dari *method* getTotalBuffer > 1024.
19. Baris 24-25 yaitu perintah untuk mencetak Proses Berhenti.
20. Baris 26 yaitu perintah untuk mencetak hasil dari *method* getduplicateStatus.
21. Baris 27-32 yaitu perulangan *if,* dimana totalBuffer > buffer maka bernilai "y" dan jika totalBuffer < buffer maka bernilai "n".
22. Baris 33 yaitu perulangan *if,* dimana mengambil nilai pada *method* getDp() dan equalIgnoreCase.
23. Baris 34-35 perintah untuk mencetak No Duplicate dan Duplicate status.
24. Baris 36-37 yaitu *method* yang digunakan untuk menutup *socket* dengan menggunakan socket.close().
25. Baris 38 yaitu perulangan *if,* dimana proses pada sisi *server* akan selesai ketika pada sisi *client* mengirimkan status *exit* dan proses akan selesai.
26. Baris 39-46 yaitu perintah untuk mencetak *Closing Connections.*

Pada tahap ini menjelaskan tentang alur *bundle* diterima oleh pihak *Intermediatte node* terlebih dahulu sebelum sampai pada pihak *client.* Pihak *intermediatte node* berperan sebagai *gateway* yaitu

menghubungkan 2 jenis jaringan komputer dengan arsitektur yang berbeda yang dihubungkan menggunakan LAN atau WAN. Namun pada penelitian ini menggunakan *interface* berupa *wifi802.11b.* Tabel 5.13 adalah *source code* untuk *Delay Tolerant Network Intermediatte Node,* yaitu:

Tabel 5. 13 *Source Code Delay Tolerant Network Intermdiatte Node*

```
1   System.out.println("INTERMEDIATTE NODE");
2           InetAddress host = InetAddress.getLocalHost();
3           Socket socket = null;
4           ObjectOutputStream oos = null;
5           ObjectInputStream ois = null;
6           for (int i = 1; i < 2; i++) {
7                               socket              =              new
    Socket(host.getHostName(), 9876);
8               oos                         =                      new
    ObjectOutputStream(socket.getOutputStream());
9               System.out.println("Connected");
10              if (i == 3) {
11                  oos.writeObject("exit");
12              } else {
13                  oos.writeObject("" + i);
14              }
15                              ois                 =                  new
    ObjectInputStream(socket.getInputStream());
16              String message = (String) ois.readObject();
17              System.out.println("Message: " + message);
18              ois.close();
19              oos.close();
20                    Thread.sleep(100);
21    public boolean getFlag() {
22          return flag;
23      }
24      public String getDp(){
25          return dp;}
26  }
```

Penjelasan:

1. Baris 1 perintah untuk mencetak *INTERMEDIATTE NODE.*
2. Baris 2-3 yaitu proses untuk mendapatkan *ip address* yang sesuai digunakan oleh *server* dengan menggunakan *method* getLocalHost().
3. Baris 4 yaitu deklarasi *socket* bernilai *null.*
4. Baris 5 proses untuk menerima *output* dari *server* dan *input* dari *server.*
5. Baris 6 perulangan *if,* dengan menggunakan *int* dimana i=1 dan i < 2.
6. Baris 7 yaitu proses menerima koneksi *socket* dari *server* dengan menggunakan *port* 9876 .
7. Baris 8 yaitu pembuatan *socket* menggunakan *method* ObjectOutputStream.
8. Baris 9 perintah untuk mencetak *connected.*
9. Baris 10-14 yaitu perulangan *if,* dimana i==3. Jika i==3 maka proses akan selesai jika tidak maka proses + i.
10. Baris 15 yaitu *server* membaca pesan atau status pesan yang terkirim dari *client* dengan menggunakan *method* socket.getInputStream.
11. Baris 16 yaitu *message* yang diterima berupa *String.*
12. Baris 17 yaitu perintah untuk mencetak *Message + message.*
13. Baris 18-19 yaitu deklarasi untuk menutup *socket.*

14. Baris 20-23 yaitu deklarasi *method* getFlag().
15. Baris 24-26 yaitu deklarasi *method* getDp().

Pada tahap ini menjelaskan tentang alur *bundle* diterima oleh pihak *client.* Pihak *client* berperan sebagai *node* yang melakukan peminjaman *bandwidth* menggunakan *Algoritme Hierarchical Token Bucket.* terdapat perbedaan antara *intermediatte node* dan *client* yaitu akan dilakukan proses *delete bundle* atau *save bundle* setelah dilakukan *duplicate bundle* pada pihak *client.* Sedagkan persamaan nya yaitu sama-sama menerima *bundle* yang dikirim oleh pihak *server.* Tabel 5.14 adalah *source code* untuk *Delay Tolerant Network* pada sisi *client,* yaitu:

Tabel 5. 14 *Source Code Bundle Delay Tolerant Network* Pada Sisi *Client*

```
1   System.out.println("CLIENT");
2           InetAddress host = InetAddress.getLocalHost();
3           Socket socket = null;
4           ObjectOutputStream oos = null;
5           ObjectInputStream ois = null;
6           for (int i = 1; i < 2; i++) {
7               // socket membuat koneksi dari server
8               socket = new Socket(host.getHostName(), 9876);
9               oos                                 =                                 new
    ObjectOutputStream(socket.getOutputStream());
10              System.out.println("Connected");
11              if (i == 3) {
12                  oos.writeObject("exit");
13              } else {
14                  oos.writeObject("" + i);
15              }
16  ois = new ObjectInputStream(socket.getInputStream());
17              String message = (String) ois.readObject();
18              System.out.println("Message: " + message);
19              ois.close();
20              oos.close();
21                      Thread.sleep(100);
22  if(total > 1024){
    System.out.println("Duplicate Total Bandwidth");
23                              duplicateStatus += "Duplicate
    Total Bandwidth";
24                          }else{
25                              System.out.println("No
    Duplicate");
26                              duplicateStatus     +=     "No
    Duplicate";
27                          }
28  if(totalBuffer > buffer){
29                              dp = "y";
30                          }
31                          else{
32                          dp = "n";
33                          }
                        dp = "y";
34                          if (dp.equalsIgnoreCase("")) {
35                              total=0;
36  System.out.println("Total sekarang Mbps " + total);
37                          }
38                          else{
39  System.out.println("Delivered Total Bandwidth");
40  }
41    public boolean getFlag() {
42          return flag;
43      }
```

```
44      public String getDp(){
45          return dp;}
46  }
```

Penjelasan:

1. Baris 1-2 yaitu proses untuk mendapatkan *ip address* yang sesuai digunakan oleh *server* dengan menggunakan *method* getocalHost().
2. Baris 3-5 yaitu deklarasi *socket* bernilai *null* dan proses untuk menerima *output* dari *server* dan *input* dari *server.*
3. Baris 6-7 perulangan *if,* dengan menggunakan *int* dimana i=1 dan i<2.
4. Baris 8 yaitu proses menerima koneksi *socket* dari *server* dengan menggunakan *port* 9876 .
5. Baris 9 yaitu pembuatan *socket* menggunakan *method* ObjectOutputStream.
6. Baris 10 perintah untuk mencetak *connected.*
7. Baris 11-15 yaitu perulangan *if,* dimana i==3. Jika i==3 maka proses akan selesai jika tidak maka proses + i.
8. Baris 16-17 yaitu *server* membaca pesan atau status pesan yang terkirim dari *client* dengan menggunakan *method* socket.getInputStream.
9. Baris 18-20 yaitu *message* yang diterima berupa *String* dan perintah mencetak *Message + message.*
10. Baris 21 yaitu deklarasi untuk menutup *socket* dengan waktu tunggu 100 detik.
11. Baris 22-27 yaitu perulangan *if,* yaitu kondisi dimana nilai total > 1024. Jika iya maka *print* "*Duplicate Total Bandwidth*" jika tidak maka *print* "*No Duplicate*"
12. Baris 28-33 perulangan *if,* dimana totalBuffer > buffer digunakan untuk "y" dan "n" digunakan untuk totalBuffer < buffer.
13. Baris 34-40 yaitu jika dp=y maka cetak total=0 dan jika tidak maka cetal Delivered Total Bandwidth.
14. Baris 41-43 yaitu deklarasi *method* getFlag().
15. Baris 44-46 yaitu deklarasi *method* getDp().

5.2 Implementasi

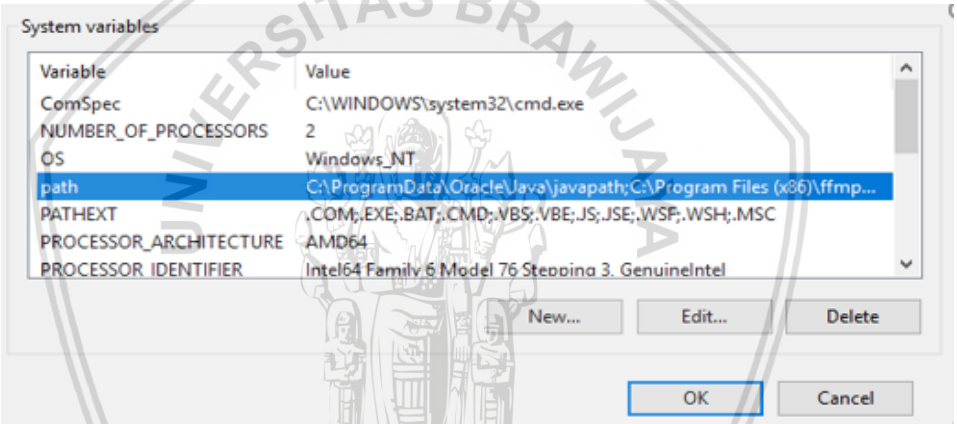
Pada tahap ini akan dilakukan proses implementasi sistem yang digunakan dalam proses penelitian. Dimulai dari proses intasalasi perangkat lunak yang digunakan untuk pengerjaan selama proses penelitian meliputi: instalasi *ONE Simulator* yang digunakan simulator dalam pengujian, *OpenJUMP* yang digunakan sebagai proses pembuatan peta pada JLS (Jalur Lintas Selatan) Malang dengan menggunakan *protocol Maxprop*. Serta pemrograman JAVA pada *Eclipse* yang digunakan sebagai implementasi *Algoritme Hierarchical Token Bucket*. Pembuatan peta pada *OpenJUMP* bisa dilakukan setelah instalasi perangkat lunak sudah berhasil dilakukan.

5.2.1 Implementasi *ONE Simulator*

ONE Simulator digunakan sebagai memodelkan pergerakan *node*, komunikasi antar *node*, *routing* dan penanganan pesan. Dengan *ONE*

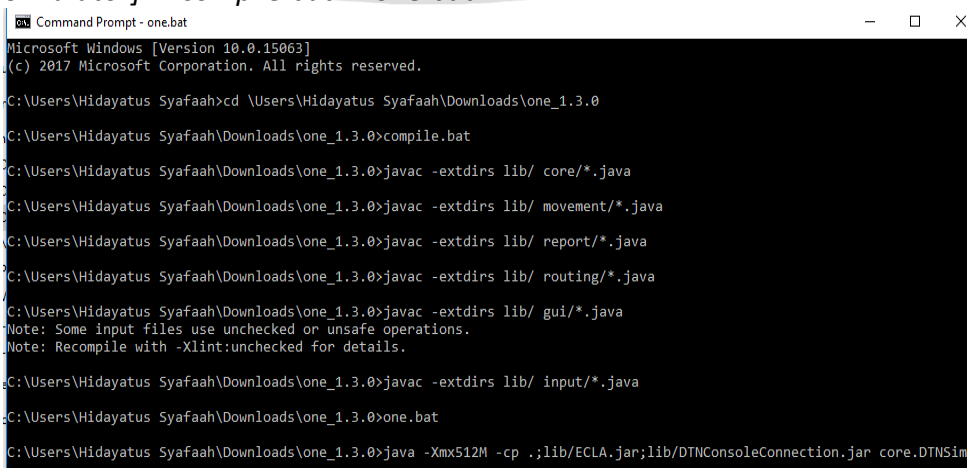
Simulator kita dapat mengetahui pergerakan antar yang kemudian digunakan untuk proses analisa terhadap *protocol MaxProp*. *ONE Simulator* adalah perangkat lunak yang berbasis *JAVA.* Jadi, sebelum melakukan proses instalasi *ONE Simulator* pastikan untuk melakukan instalasi *JAVA* pada komputer yang akan digunakan untuk melakukan implementasi. Pastikan bahwa *path* pada komputer yang digunakan sudah *disetting*, hal ini ditujukan untuk mengetahui letak JAVA yang sudah diinstal. Berikut adalah proses *setting* pada *path* komputer yang digunakan:

Pertama pilih *Control Panel-> View->Advance System Setting->Environment Variables -> System Variable -> Path ->Edit.* Pada *file edit* diganti dengan nama file yang sesuai dengan tempat letaknya *JDK (Java Development Kit)* yang digunakan untuk instalasi *ONE Simulator*. Pada bagian ini, *jdk* diletakkan di *C:\Users\Hidayatus Syafaah\Downloads\Documents* versi 1.8.0. Gambar 5.7 menunjukan proses *setting path* pada komputer yang digunakan.



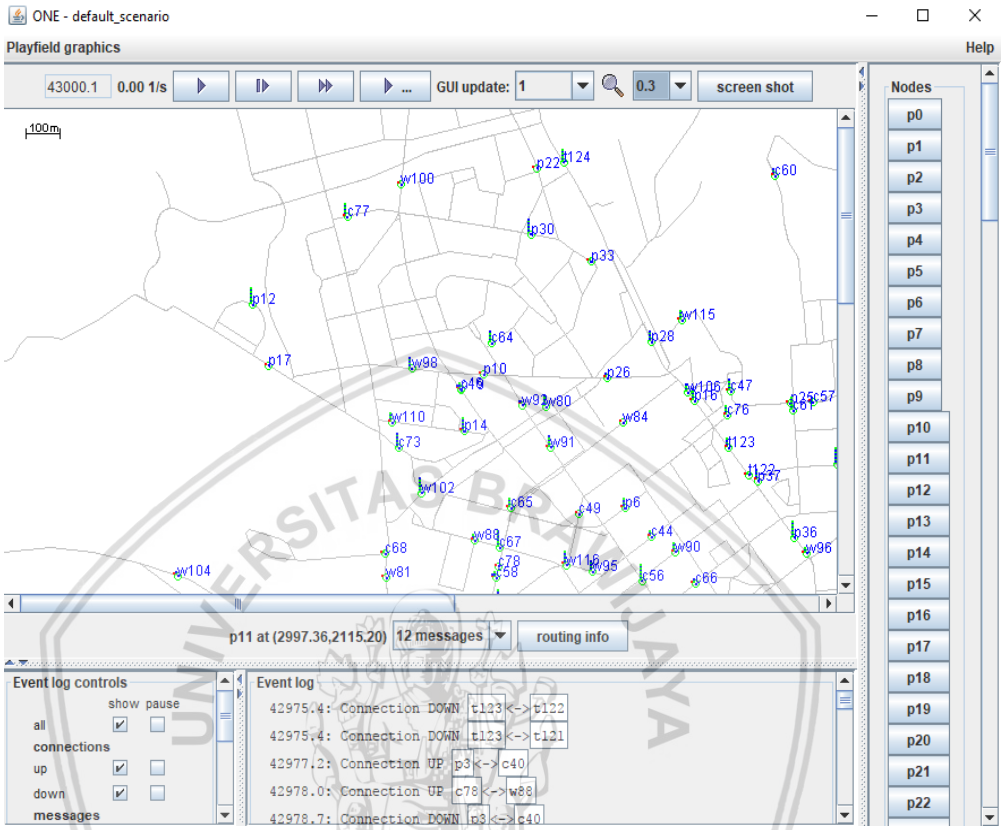
Gambar 5. 7 *Setting Path JDK*

Setelah *setting path* selesai dilakukan kita bisa melakukan *run* pada *ONE Simulator* dengan menggunakan *command prompt* dengan perintah *cd \Users\Hidayatus Syafaah\Pictures\one_1.4.1 [letak file ONE Simulator] -> compile.bat -> one.bat.*



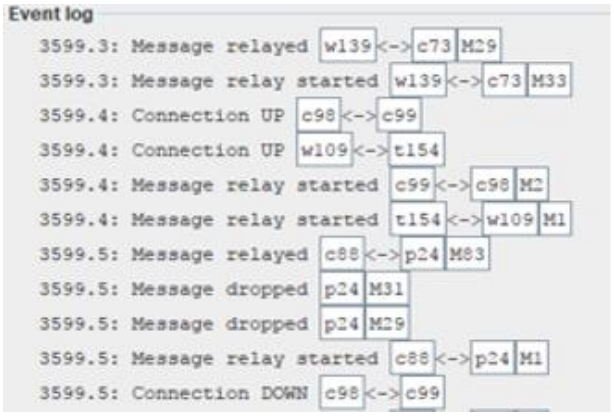
Gambar 5. 8 *Compile and Run* in *ONE Simulator*

Setelah proses *compile* selesai maka *ONE Simulator* akan menampilkan antarmuka seperti dibawah ini.



Gambar 5. 9 Tampilan *Default Setting ONE Simulator*
Sumber : [(Pengujian)]

Dari gambar 5.9 bisa diketahui berapa banyak jumlah *node* yang digunakan dalam proses simulasi. Pada gambar diatas bisa dilihat bahwa waktu yang digunakan selama proses simulasi yaitu 43000s atau 12h. Ketika kita menjalankan *ONE Simulator* kita dapat mengetahui apakah sebuah pesan di *drop* ketika pengiriman pesan sedang terjadi. Gambar 5.10 menunjukan proses pengiriman pesan yang berlangsung:

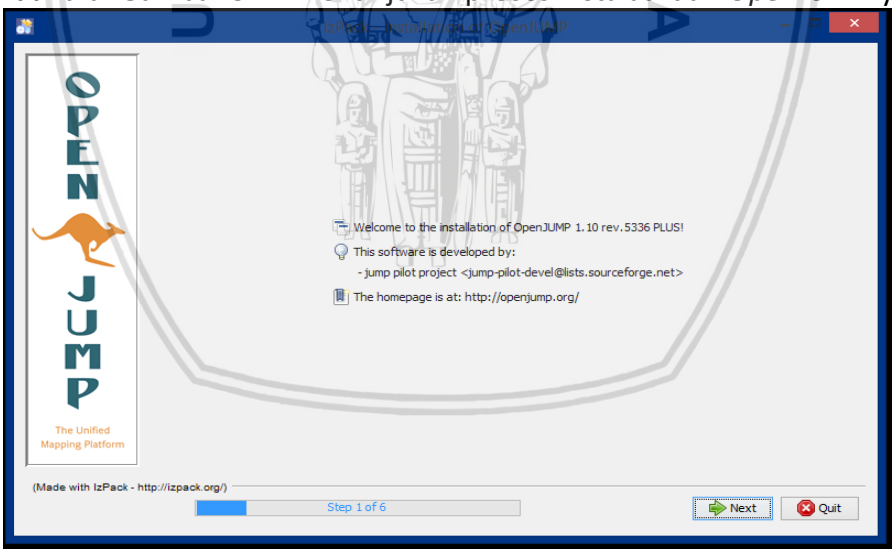


Gambar 5. 10 Pengiriman Pesan antar *node*

Dari hasil gambar 5.10 bisa dilihat bahwa, terdapat beberapa status pengiriman pesan antar *node* yang sedang terjadi seperti *message relayed, message relay started, connection UP, message dropped* dan *connection DOWN.*

5.2.2 Implementasi *OpenJUMP*

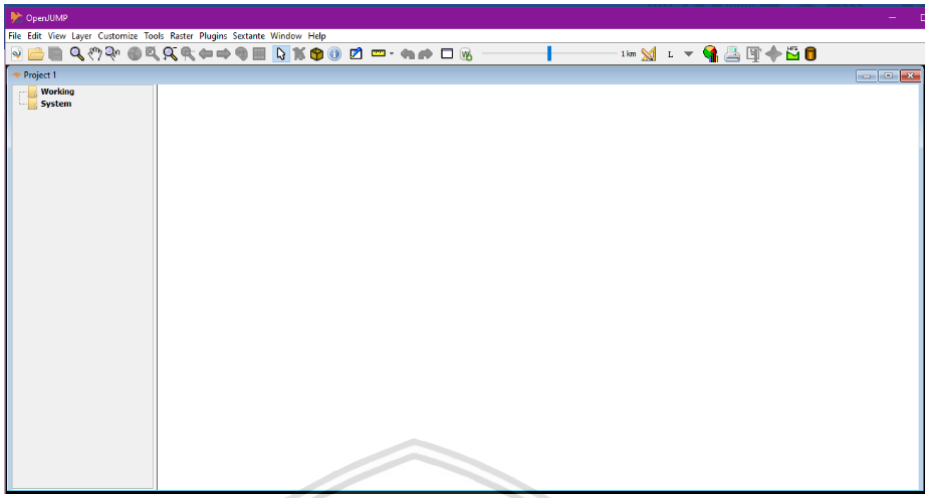
OpenJUMP digunakan untuk membuat peta yang akan digunakan dalam penelitian ini. *OpenJUMP* adalah perangkat lunak yang berbasis JAVA sehingga dalam proses instalasinya harus diinstall JAVA terlebih dahulu. Gambar 5.11 menunjukan proses instalasi dari *OpenJUMP* yaitu:



Gambar 5. 11 Proses Instalasi *OpenJUMP*

Setelah instalasi selesai dilakukan, kita bisa melihat tampilan dari

antarmuka *OpenJUMP*.

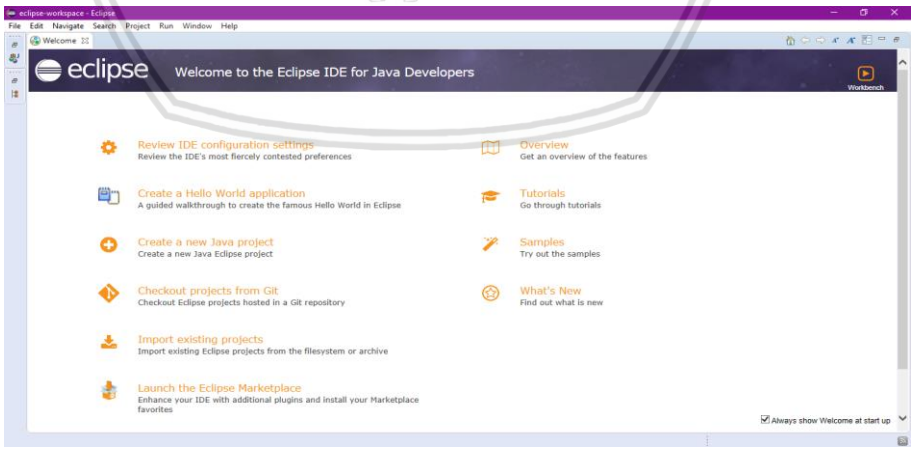


Gambar 5. 12 Gambar Antar Muka *OpenJump*

Dari antarmuka diatas kita bisa membuat peta yang akan dijadikan sebagai daerah simulasi yang sesuai dengan peta aslinya dengan cara pilih *menu file -> new -> project* atau melakukan *import file* peta secara langsung.

5.2.3 Implementasi JAVA pada *Eclipse*

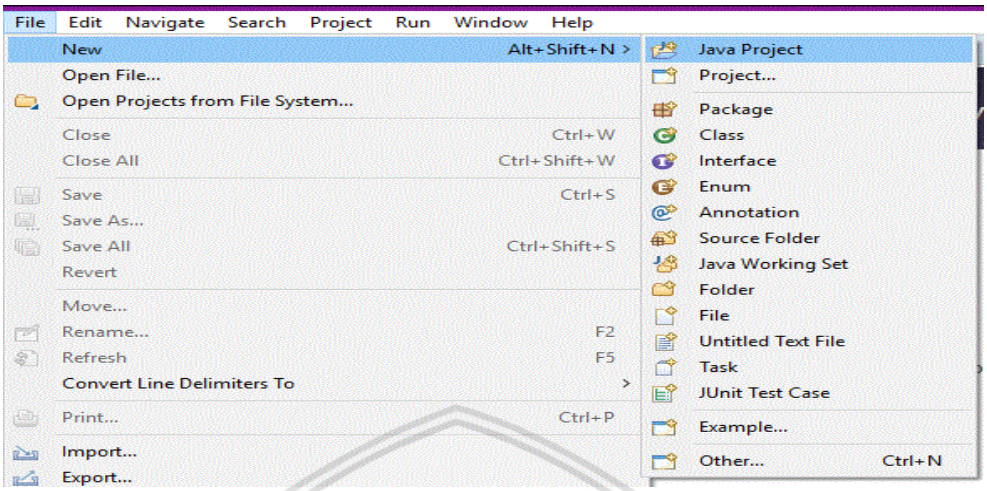
Bahasa pemrograman *JAVA* pada *Eclipse* yang digunakan untuk melakukan implementasi *Algoritme Hierarchical Token Bucket* yang akan digunakan pada penelitian ini. Bahasa pemrograman *JAVA* adalah bahasa pemrograman tingkat tinggi yang berorientasi objek dan tersusun dari beberapa bagian yang disebut klas. Gambar 5.13 menunjukan tampilan antarmuka *JAVA* pada *Eclipse* yang sudah terpasang pada komputer yang akan digunakan yaitu:



Gambar 5. 13 Antar Muka JAVA Pada *Eclipse*

Pada penelitian ini, *source code* dari pemrograman *JAVA* dengan menerapkan *Algoritme Hierarchical Token Bucket* akan di masukkan pada simulator yang akan digunakan yaitu *ONE Simulator*. Berikut langkah-langkahnya yaitu:

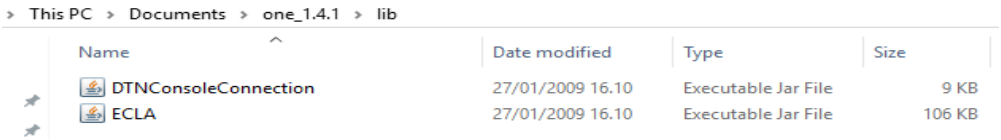
1. Langkah pertama yaitu, membuat *project* baru pada *Eclipse,* dimulai dengan klik menu *file* pada *Eclipse* kemudian pilih *new – Java Project.*



Gambar 5. 14 Proses Pembuatan *Project* Baru Pada *Eclipse*

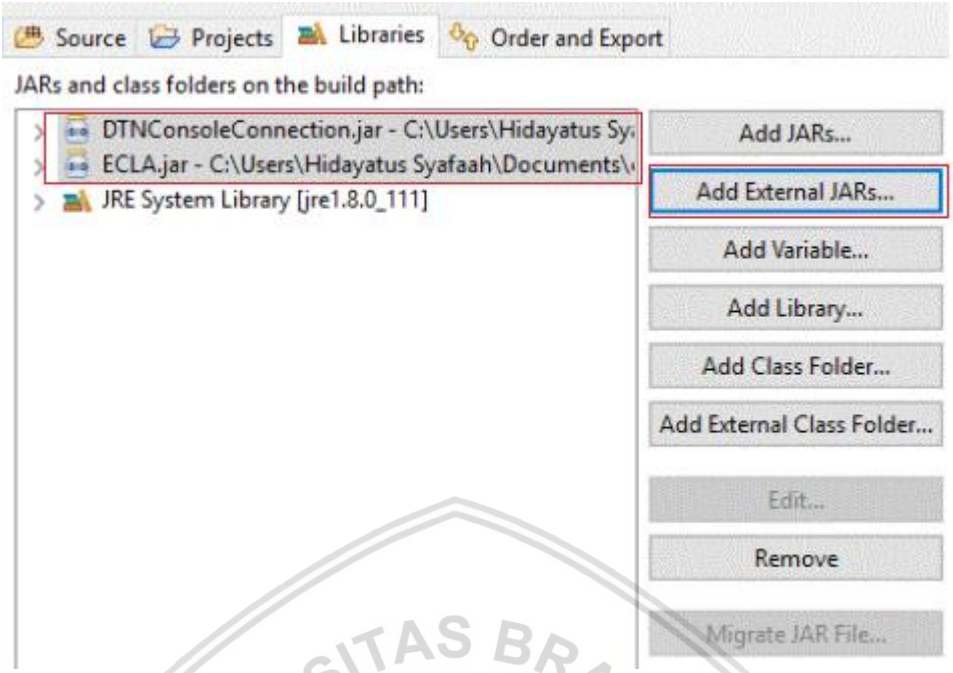
2. Langkah kedua yaitu mendefinisikan nama *project* yang akan dibuat. Seperti pada penelitian ini dengan mengunakan nama *ONE.* Berikut adalah tampilan dari nama pembuatan *project* pada *Eclipse.* Pada tahap ini kita perlu menghapus tanda centang pada *Use Default Location* supaya kita bisa mengimport *folder ONE Simulator* yang akan digunakan. Kemudian kita pilih *menu browse* yang bertujuan untuk mengimport *folder ONE Simulator.* Seperti pada penelitian ini menggunakan *ONE Simulator versi 1.4.1* atau *versi* terbaru dari 1.3.0 RC setelah itu kita memilih tombol *next* untuk masuk ke *libraries* konfigurasi.

3. Langkah yang ketiga yaitu mengimport dua *file* penting dengan *ekstensi* .jar pada *folder ONE Simulator* yaitu *DTNConsoleConnection.jar* dan *ECLA.jar.* Langkah-langkahnya yaitu klik *menu librarie – add external jar* kemudian pilih *DTNConsoleConnection.jar* dan *ECLA.jar* pada *ONE Simulator* yang terletak didalam *folder lib* setelah itu klik *finish* untuk menyelesaikan proses konfigurasipada*Eclipse*.



Gambar 5. 15 *DTN Console Connection* Dan *ECLA File*

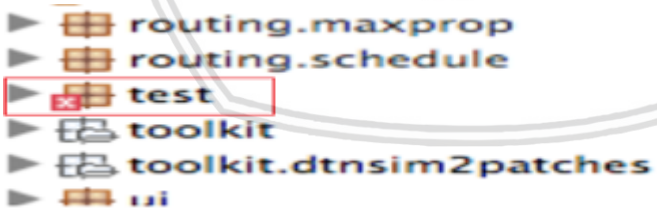
File DTNConsoleConnection dan *ECLA.jar* merupakan *file* bawaan dari *ONE Simulator* yang akan digunakan. Kedua *file* ini digunakan untuk mengimport semua *folder* yang ada pada *ONE Simulator* yang akan dimasukkan pada *Eclipse.*



Gambar 5. 16 *Import DTNConsoleConnection.jar* dan *ECLA.jar*

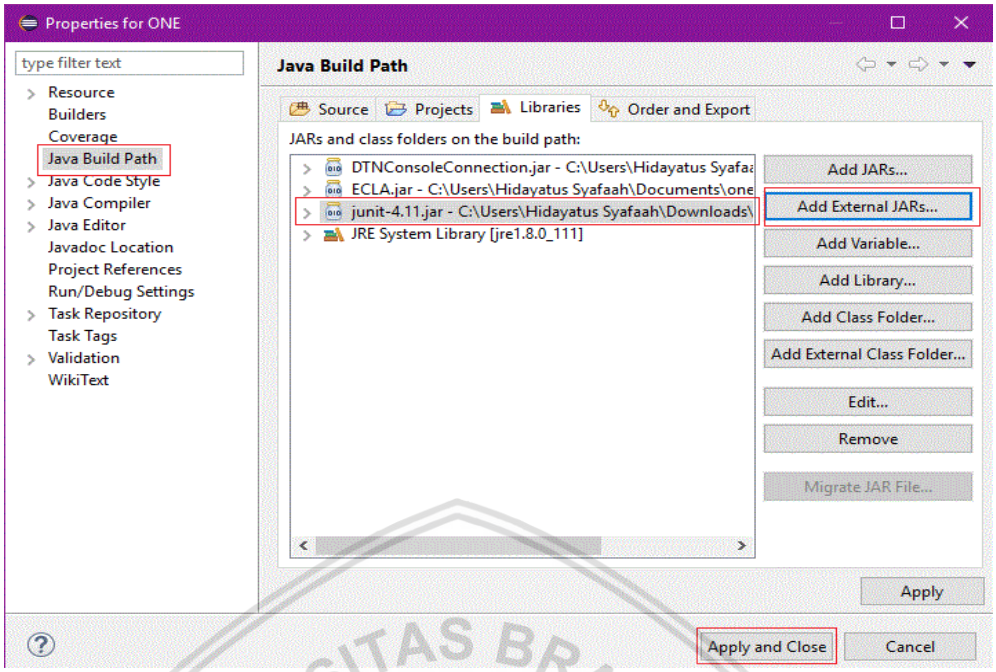
Setelah *file DTNConsoleConnection* dan *ECLA.jar* selesai di *import,* maka *project* yang kita buat bisa kita lihat pada tampilan antarmuka *Eclipse.*

4. Setelah proses konfigurasi selesai, maka *project* yang kita buat bisa dilihat sudah terdaftar pada *Eclipse* seperti pada gambar 5. Namun ada kesahalan pada paket *test* maka ketika kita akan men-debug aplikasi yang sudah dibuat, maka kita harus menghapus paket *test* agar terhindar dari kesalahan saat kita menjalankan aplikasi tersebut. Menghapus paket *test* ini tidak akan mempengaruhi fitur dan fungsi pada *ONE Simulator.* Proses untuk memperbaiki paket *test* bisa kita lihat pada gambar dibawah ini:



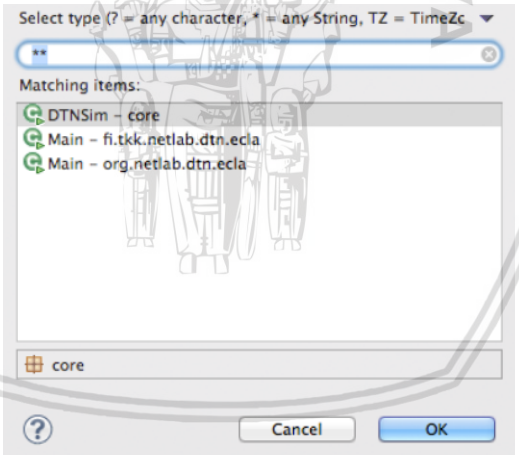
Gambar 5. 17 Tampilan *Folder ONE Simulator* Yang Sudah Dipasang Pada *JAVA Eclipse*

5. Paket *test* adalah *test case* untuk berbagai fungsi dan kelas dalam *ONE Simulator*. Menghapus paket ini tidak mempengaruhi fungsi pada *ONE Simulator.* Kemudian klik \ *properties,* setelah itu, kita pilih *Java Build Path* dan kemudian pilih \ *Libraries*. Kemudian klik \ *add external jar* dan *import jar Junit* yang sudah didownload kemudian pilih *apply and close.* Setelah kita mengimpor *jar* ini, kesalahan akan hilang.



Gambar 5. 18 Proses *Import Junit-4.11.jar*

6. Setelah *error* hilang, kita bisa menjalankan *project* yang sudah dibuat pada *Eclipse* dengan menggunakan *ONE Simulator* seperti langkah-langkah seperti klik *run* – pilih *Java Application* kemudia pilih *DTNSim - Core.*



Gambar 5. 19 Proses *Running* Pada *JAVA Eclipse* Dengan Memilih *DTNSim.java*

Penjelasan:

- Pada *ONE Simulator* menggunakan dua metode untuk menjalankan simulator ini. Yang pertama yaitu berdasarkan UI yaitu dengan memanggil fungsi *DTNSim.java* yang terdapat dalam *folder ONE Simulator "\core"* yang berfungsi sebagai *main* dalam *ONE Simulator* dan yang kedua didasarkan pada terminal, yaitu *print out commands* dan *alert* informasi di terminal tanpa antarmuka pengguna yang disebut *mode batch*.

- Kemudian, *DTNSim.java* akan memanggil *function* *initSettings*. Fungsi ini akan membaca nilai dari *default settings.txt* yang sudah dikonfigurasi sebelumnya yang mengacu pada tabel 4.1.
- Selanjutnya, berdasarkan parameter *DTNSim.java* akan memanggil *DTNSimGUI.java* untuk memulai skenario dan mensimulasikan jaringan. DTN. *DTNSimGUI* adalah *subclass* dari *DTNSimUI*.
- Ketika dua *node* saling bertemu dan terhubung pada *file* konfigurasi berdasarkan *setting range*, *network interface* maka *ActiveRouter.java* atau *PassiveRouter.java* akan dipanggil. Kedua *router* tersebut merupakan kelas induk *routing*. Ketika kita menerapkan *algoritme routing*, maka perlu menggunakan salah satu dari dua kelas *routing* tersebut. Dimana *EpedemicRouter* mewarisi *ActiveRouter.java* pada *ONE Simulator*.
- Pada *ONE Simulator ActiveRouter.java* diwarisi dari *MessageRouter.java. MessageRouter.java* memiliki tanggung jawab untuk mengelola *buffer* menyimpan dan mengelola pesan selama proses pergerakan *node*.

BAB 6 PENGUJIAN DAN HASIL

6.1 Hasil Pengujian *Protocol MaxProp*

Hasil pengujian didapat dari pengujian terhadap *protocol routing maxprop* dengan jumlah *node* dari 50, 100, 150 dan 200 dengan kecepatan dimulai dari 20-160 km/jam dan ukuran pesan 1 MB. Jumlah *node* ini dipilih berdasarkan jumlah maksimal pengunjung pada daerah simulasi Jalur Lintas Selatan Malang untuk 1 minggu Berikut ini hasil yang diperoleh untuk setiap parameter pengujian yang dilakukan seperti:

6.1.1 *Average Latency*

Pengujian akan dilakukan dengan menerapkan setiap jumlah *node* dan kecepatan *node* yang berbeda pada *protocol MaxProp*. Hasil dari pengujian *average latency* bisa dilihat dari tabel 6.1 berikut ini:

Tabel 6. 1 Nilai *Average Latency* Untuk *Protocol MaxProp*

Protocol Routing	Jumlah *node*	Hasil Pengujian *Average Latency* (ms)		
MaxProp	50 *node*	1134.8947ms	1147.5100ms	1276.5700ms
	100 *node*	1072.5196ms	1078.2040ms	1089.4194ms
	150 *node*	1058.6030ms	1231.7296ms	1029.8541ms
	200 *node*	1074.9700ms	821.3750ms	220.2333ms

Pada hasil pengujian tabel 6.1 menggunakan *map route movement* sebagai *movement model* dimana model pergerakan ini hanya mengikuti jalur yang sudah ditentukan oleh peta, *buffer size* yang digunakan sebesar 1GB. Dimana waktu tunggu yang digunakan adalah minimal 0s dan maxsimal 120s, waktu simulasi 3600s, map file yang digunakan untuk *protocol routing maxprop dan Algoritme Hierarchical Token Bucket* yaitu Peta.wkt dan area simulasi yang digunakan yaitu 5000 * 5000. Dimana area yang digunakan menyesuaikan dengan ukuran asli pada daerah uji yaitu 25Km. Kecepatan yang digunakan untuk hasil pengujian pada tebel 5.1 adalah 20-40 km/jam dengan ukuran pesan 1Mb pada setiap jumlah *node* yang berbeda-beda yaitu 50, 100 , 150 dan 200 *node*. Jumlah *node* ini dipilih berdasarkan jumlah maksimal pengunjung pada daerah simulasi Jalur Lintas Selatan Malang untuk 1 minggu

6.1.2 *Overhead Ratio*

Pada pengujian *overhead ratio,* diperoleh dari hasil pengujian *protocol MaxProp* dengan jumlah node antara 50, 100, 150 dan 200 *node* dan kecepatan *node* 20-40, 40-80, 80-160 km/jam dan ukuran pesan sebesar 1 MB. Jumlah *node* ini dipilih berdasarkan jumlah maksimal pengunjung pada daerah simulasi Jalur Lintas Selatan Malang untuk 1 minggu Hasil dari pengujian *overhead ratio* bisa dilihat dari tabel 6.2 dibawah ini:

Tabel 6. 2 Nilai *Overhead Ratio* Untuk *Algoritme Hierarchical Token Bucket*

Protocol Routing	Jumlah *node*	Hasil Pengujian *Overhead Ratio*(ms)		
MaxProp	50 *node*	15.9737ms	61.9000ms	84.3000ms
	100 *node*	42.5536ms	68.8000ms	74.0278ms
	150 *node*	77.2090ms	116.4074ms	144.9189ms
	200 *node*	112.0571ms	168.7500ms	158.1667ms

Hasil pengujian tabel 6.2 diperoleh dari implementasi *protocol MaxProp.* Dengan menggunakan beberapa jumlah *node* yang berbeda yaitu 50, 100, 150 *node* dan 200 *node* dengan kecepatan *node* antara 20-40km/jam, 40-80km/jam dan 80-160km/jam. Jumlah *node* ini dipilih berdasarkan jumlah maksimal pengunjung pada daerah simulasi Jalur Lintas Selatan Malang untuk 1 minggu

6.1.3 *Delivery Probability*

Pada pengujian *delivery probability* diperoleh dari hasil pengujian *protocol MaxProp* dengan jumlah node antara 50, 100, 150 dan 200 *node* dan kecepatan node 20-40, 40-80, 80-160 km/jam dan ukuran pesan sebesar 1 MB. Jumlah *node* ini dipilih berdasarkan jumlah maksimal pengunjung pada daerah simulasi Jalur Lintas Selatan Malang untuk 1 minggu Hasil dari pengujian *delivery probability* bisa dilihat dari tabel 6.3 dibawah ini:

Tabel 6. 3 Nilai *Delivery Probability* Untuk *Protocol MaxProp*

Protocol Routing	Jumlah *node*	Hasil Pengujian *Delivery Probability*(%)		
MaxProp	50 *node*	0.3115%	0.0820%	0.0820%
	100 *node*	0.4628%	0.2066%	0.2975%
	150 *node*	0.5537%	0.2231%	0.3058%
	200 *node*	0.5738%	0.2025%	0.1714%

Pengujian dilakukan dengan menerapkan jumlah *node* yang sudah ditentukan untuk pengujian dengan kecepatan 20-40km/jam, 40-80km/jam dan 80-160km/jam. Jumlah *node* yang digunakan berdasarkan jumlah pengunjung pada daerah uji Jalur Lintas Selatan Malang yaitu antara 50-200 pengunjung.

6.1.4 *Average Hop Count*

Pada pengujian *average hop count* diperoleh dari hasil pengujian *protocol MaxProp* dengan jumlah node antara 50, 100, 150 dan 200 *node* dan kecepatan node 20-40, 40-80, 80-160 km/jam dan ukuran pesan sebesar 1 MB. Jumlah *node* ini dipilih berdasarkan jumlah maksimal pengunjung pada

daerah simulasi Jalur Lintas Selatan Malang untuk 1 minggu Hasil dari pengujian *average hop count* bisa dilihat dari tabel 6.4 dibawah ini:

Tabel 6. 4 Nilai *Average Hop Count* Untuk *Protocol MaxProp*

Protocol Routing	Jumlah *node*	Hasil Pengujian *Average Hop Count*(ms)		
MaxProp	50 *node*	2.2368ms	2.6000ms	2.9000ms
	100 *node*	2.8929ms	3.8400ms	5.1944ms
	150 *node*	3.2985ms	5.2222ms	8.0811ms
	200 *node*	3.7143ms	5.0625ms	3.6667ms

Pada hasil pengujian tabel 6.4 menggunakan *map route movement* sebagai *movement model* dimana model pergerakan ini hanya mengikuti jalur yang sudah ditentukan oleh peta, *buffer size* yang digunakan sebesar 1GB, waktu simulasi 3600s, map file yang digunakan untuk *protocol routing maxprop* yaitu Peta.wkt dan area simulasi yang digunakan yaitu 5000*5000 dengan ukuran pesan 1Mb pada setiap jumlah *node* yang berbeda-beda yatu 50, 100, 150 dan 200 *node*. Jumlah *node* ini dipilih berdasarkan jumlah maksimal pengunjung pada daerah simulasi Jalur Lintas Selatan Malang untuk 1 minggu

6.2 Hasil Pengujian *Algoritme Hierarchical Token Bucket*

Hasil pengujian pada tahap ini diperoleh dari *Algoritme Hierarchical Token Bucket* dengan melakukan seleksi *node routing multi copy*. Pengujian yang dilakukan dengan jumlah *node* 50, 100, 150 dan 200 *node*, kecepatan *node* 20-160km/jam dengan ukuran pesan 1Mb. Dari hasil pengujian akan diperoleh nilai untuk mengetahui jumlah *Average Latency, Overhead Ratio, Delivery Probability* dan *Average Hop Count.* Jumlah *node* ini dipilih berdasarkan jumlah maksimal pengunjung pada daerah simulasi Jalur Lintas Selatan Malang untuk 1 minggu

6.2.1 Average Latency

Pada pengujian *average latency,* diperoleh dari hasil pengujian *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy* dengan jumlah node 50, 100, 150 dan 200 *node* dan kecepatan *node* 20-40, 40-80, 80-160 km/jam dan ukuran pesan sebesar 1 MB. Jumlah *node* ini dipilih berdasarkan jumlah maksimal pengunjung pada daerah simulasi Jalur Lintas Selatan Malang untuk 1 minggu. Hasil dari pengujian *average latency* bisa dilihat dari table 6.5 dibawah ini:

Tabel 6. 5 Nilai *Average Latency* Untuk *Algoritme Hierarchical Token Bucket*

Protocol Routing	Jumlah *node*	Hasil Pengujian *Average*

		Latency(ms)		
MaxProp	50 *node*	980.0307ms	544.9934ms	327.2009ms
	100 *node*	754.4082ms	427.1321ms	239.1272ms
	150 *node*	643.7371ms	358.3009ms	232.6261ms
	200 *node*	522.0510ms	324.8652ms	220.4000ms

Hasil pengujian tabel 6.5 diperoleh dari pengujian *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy.* Dengan menggunakan *map route movement* sebagai model pergerakannya dan *buffer* sebesar 1GB untuk menampung pesan yang dikirim. *Buffer* ini dipilih berdasarkan banyaknya jumlah *node* yang digunakan yaitu 50, 100, 150 *node* dan 200 *node.* Jumlah *node* ini dipilih berdasarkan jumlah maksimal pengunjung pada daerah simulasi Jalur Lintas Selatan Malang untuk 1 minggu.

6.2.2 Overhead Ratio

Pada pengujian *overhead ratio* diperoleh dari hasil pengujian *Algoritme Hierarchical Token Bucket* dengan jumlah node antara 50, 100, 150 dan 200 *node* dan kecepatan node 20-40, 40-80, 80-160 km/jam dan ukuran pesan sebesar 1 MB. Jumlah *node* ini dipilih berdasarkan jumlah maksimal pengunjung pada daerah simulasi Jalur Lintas Selatan Malang untuk 1 minggu. Hasil dari pengujian *overhead ratio* bisa dilihat dari tabel 6.6 dibawah ini:

Tabel 6. 6 Nilai *Overhead Ratio* Untuk *Algoritme Hierarchical Token Bucket*

Protocol Routing	Jumlah *node*	Hasil Pengujian *Overhead Ratio*(ms)		
MaxProp	50 *node*	48.6705ms	45.1038ms	38.7876ms
	100 *node*	136.2268ms	135.0917ms	119.6491ms
	150 *node*	438.9691ms	346.9091ms	249.4696ms
	200 *node*	898.2000ms	668.6429ms	482.7913ms

Pada hasil pengujian tabel 6.6 menggunakan *map route movement* sebagai *movement model* dimana model pergerakan ini hanya mengikuti jalur yang sudah ditentukan oleh peta, *buffer size* yang digunakan sebesar 1GB, waktu simulasi 3600s, map file yang digunakan untuk *protocol maxprop dan Algoritme Hierarchical Token Bucket* yaitu Peta.wkt dan area simulasi yang digunakan yaitu 5000*5000 dengan ukuran pesan 1MB pada setiap jumlah *node* yang berbeda-beda yaitu 50, 100, 150 dan 200 *node.* Jumlah *node* ini dipilih berdasarkan jumlah maksimal pengunjung pada daerah simulasi Jalur Lintas Selatan Malang untuk 1 minggu.

6.4.3 Delivery Probability

Pada pengujian *delivery probability,* diperoleh dari hasil implementasi *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy.*

dengan jumlah node antara 50, 100, 150 dan 200 *node* dan kecepatan *node* 20-40, 40-80, 80-160 km/jam dan ukuran pesan sebesar 1 MB. Jumlah *node* ini dipilih berdasarkan jumlah maksimal pengunjung pada daerah simulasi Jalur Lintas Selatan Malang untuk 1 minggu Hasil dari pengujian *delivery probability* bisa dilihat dari tabel 6.7 dibawah ini:

Tabel 6. 7 Nilai *Delivery Probability* Untuk *Algoritme Hierarchical Token Bucket*

Protocol Routing	Jumlah *node*	Hasil Pengujian *Delivery probability*(%)		
MaxProp	50 *node*	0.7213%	0.8689%	0.9262%
	100 *node*	0.8017%	0.9008%	0.9421%
	150 *node*	0.8017%	0.9091%	0.9504%
	200 *node*	0.8197%	0.9180%	0.9426%

Hasil pengujian tabel 6.7 diperoleh dari implementasi *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy.* Dengan menggunakan jumlah *node* yang berbeda yaitu 50, 100, 150 *node* dan 200 *node* dengan kecepatan *node* antara 20-40km/jam, 40-80km/jam dan 80-160km/jam. Jumlah *node* ini dipilih berdasarkan jumlah maksimal pengunjung pada daerah simulasi Jalur Lintas Selatan Malang untuk 1 minggu.

6.4.4 *Average Hop Count*

Pada pengujian *average hop count,* diperoleh dari hasil pengujian *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy* jumlah node antara 50, 100, 150 dan 200 *node* dan kecepatan *node* 20-40, 40-80, 80-160 km/jam dan ukuran pesan sebesar 1 MB. Jumlah *node* ini dipilih berdasarkan jumlah maksimal pengunjung pada daerah simulasi Jalur Lintas Selatan Malang untuk 1 minggu. Hasil dari pengujian *average hop count* bisa dilihat dari tabel 6.8 dibawah ini:

Tabel 6. 8 Nilai *Average Hop Count* Untuk *Algoritme Hierarchical Token Bucket*

Protocol Routing	Jumlah *node*	Hasil Pengujian *Average Hop Count*(ms)		
MaxProp	50 *node*	2.9205ms	3.0566ms	2.8407ms
	100 *node*	5.5670ms	5.2202ms	4.2018ms
	150 *node*	12.5258ms	9.2727ms	6.6174ms
	200 *node*	20.5900ms	12.8571ms	7.7565ms

Hasil pengujian tabel 6.8 diperoleh dari pengujian *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy.* Dengan menggunakan *map route movement* sebagai model pergerakannya dan *buffer* sebesar 1GB untuk menampung pesan yang dikirim. *Buffer* ini dipilih berdasarkan

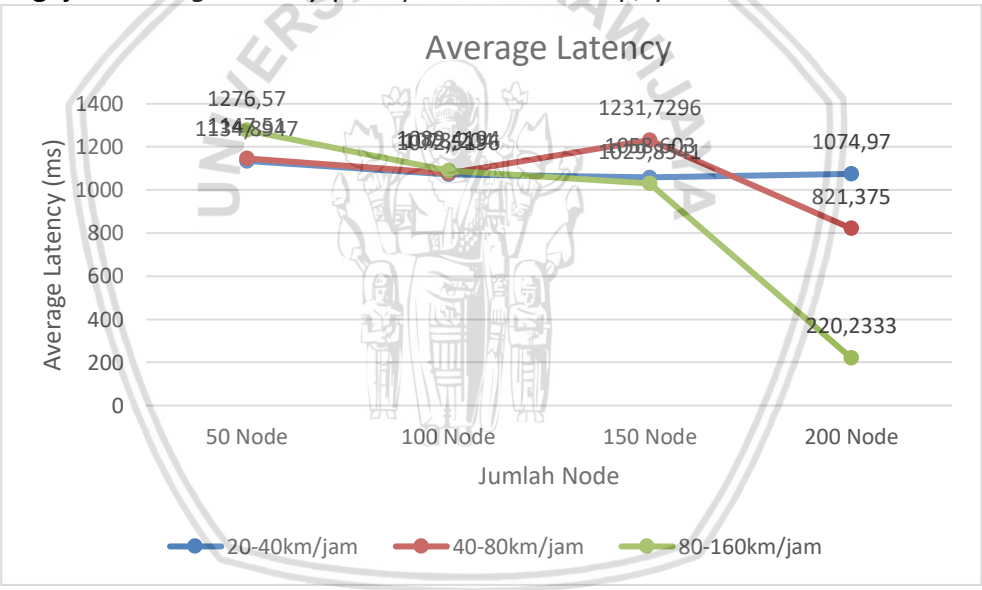
banyaknya jumlah *node* yang digunakan yaitu 50, 100, 150 *node* dan 200 *node.* Pesan yang dikirimkan sebesar 1MB dengan menggunakan beberapa kecepatan *node* seperti 20-40km/jam, 40-80km/jam dan 80-160km/jam. Jumlah *node* ini dipilih berdasarkan jumlah maksimal pengunjung pada daerah simulasi Jalur Lintas Selatan Malang untuk 1 minggu.

6.3 Pembahasan *Protocol MaxProp*

Pada tahap ini, pembahasan dilakukan setelah pengujian untuk *protocol MaxProp* Hasil pengujian yang didapat berupa nilai *average latency, overhead ratio, delivery probability dan average hop count* yang akan ditampilkan dalam bentuk grafik agar mempermudah peneliti dalam melakukan proses analisi.

6.3.1 Pembahasan Hasil Pengujian *Average Latency*

Pada tahap ini dilakukan perbandingan untuk setiap pengujian *average latency* dengan jumlah *node* 50, 100, 150 *node* dan 200 *node* dengan kecepatan *node* yang berbeda-beda. Gambar 6.1 menunjukan grafik dari pengujian *average latency* pada *protocol MaxProp,* yaitu:



Gambar 6. 1 Hasil Pengujian *Average Latency* Untuk *Protocol MaxProp*

Hasil pengujian *average latency* cenderung naik turun untuk setiap penambahan jumlah *node*nya. Jumlah *node* yang digunakan yaitu 50, 100, 150 dan 200 *node* dengan kecepatan 20-40, 40-80 dan 80-160 km/jam dengan ukuran *buffer* sebesar 1GB. Nilai *average latency* dipengaruhi oleh ukuran *buffer* yang digunakan yaitu semakin besar ukuran *buffer* maka nilai untuk *average latency* cenderung mengalami penurunan. Hal ini disebabkan oleh penentuan prioritas pesan yang dikirim pada *protocol MaxProp* yang bergantung pada nilai *threshold* pada *buffer*.

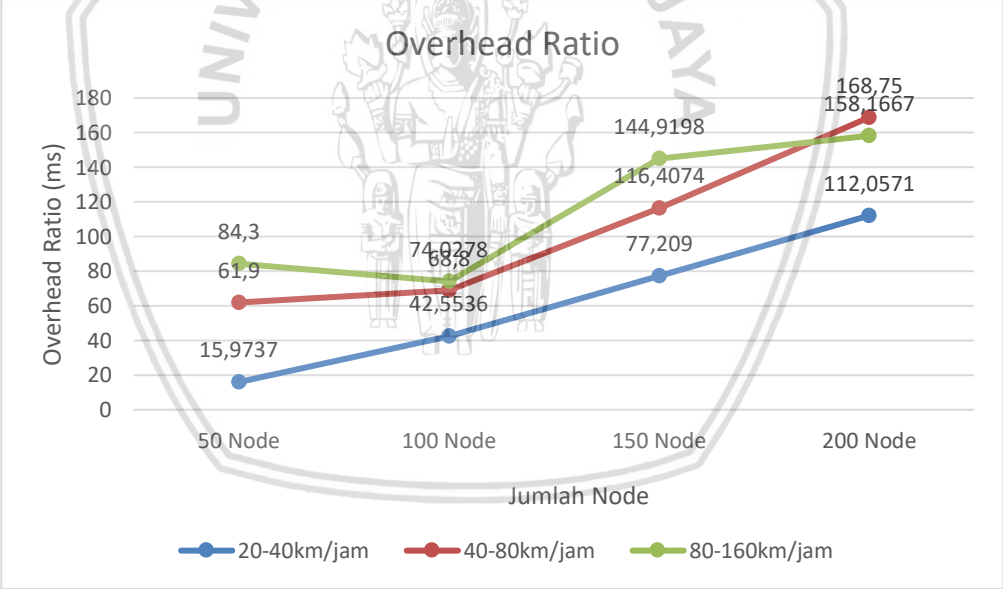
Ketika jumlah *node* 50 nilai *average latency* cenderung naik yaitu 1134.8947ms, hal ini disebabkan karena dengan semakin besar ukuran *buffer* maka pesan yang ditampung akan semakin banyak. Sedangkan dengan jumlah *node* yang sedikit, kesempatan pengiriman sedikit dan pesan

yang dikirimkan dilihat dari prioritas nilai *hopcount* terkecil. Ketika jumlah *node* ditambah menjadi 100, 150 dan 200 *node* nilai *average latency* yang diperoleh untuk kecepatan *node* 20-40 km/jam mengalami penurunan jumlah yaitu 1072.5196ms 1058.6030ms dan 1074.9700ms. Hal ini disebabkan oleh probabilitas suatu *node* untuk bertemu dengan *node* yang lainnya kecil. Ketika sebuah *node* memiliki kesempatan untuk mengirimkan pesan, maka pesan akan dikirimkan sesuai dengan prioritasnya. Semakin kecil bertemunya suatu *node* dengan *node* yang lain, maka saat *node* memiliki kesempatan untuk mengirimkan pesan akan langsung dikirimkan karena sedikitnya pesan dalam antrian dan nilai *hop count* setiap pesan kecil.

Pada *Maxprop* pengiriman pesan dipengaruhi oleh nilai *hopcount* dan pergerakan *node*. Nilai latency kecil apabila *node* sumber dengan *node* tujuan sering bertemu secara langsung. Akan tetapi nilai *average latency* tetap cenderung turun ketika nilai *buffer* dinaikkan.

6.3.2 Pembahasan Hasil Pengujian *Overhead Ratio*

Pada tahap ini dilakukan perbandingan untuk setiap pengujian *overhead ratio* dengan jumlah *node* 50, 100, 150 dan 200 *node* dengan kecepatan *node* yang berbeda-beda. Gambar 6.2 menunjukan ini menunjukkan hasil pengujian pada *protocol MaxProp,* yaitu:



Gambar 6. 2 Hasil Pengujian *Overhead Ratio* Untuk *Protocol MaxProp*

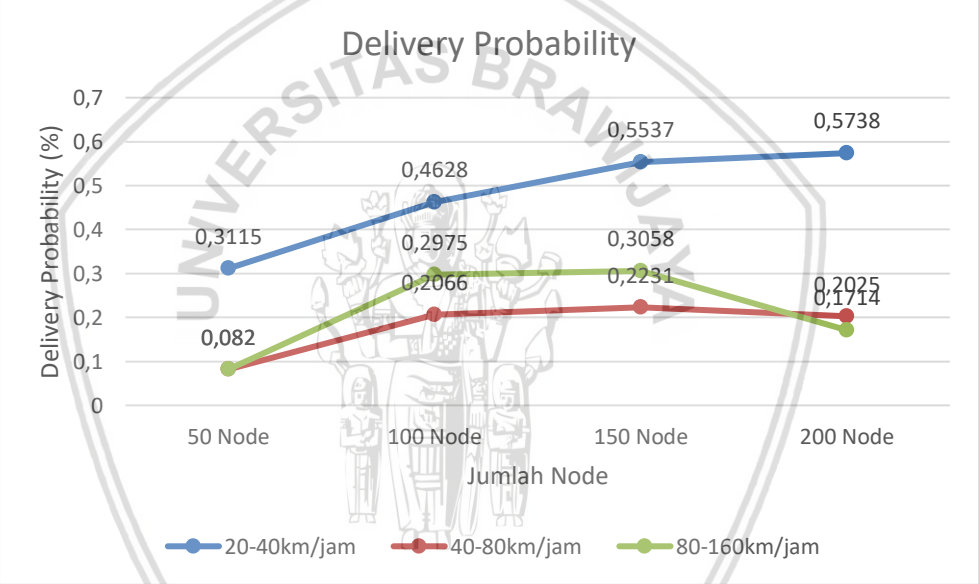
Pada hasil pengujian *protocol MaxProp* ketika berada pada kecepatan *node* 20-40km/jam untuk jumlah *node* 50, 100, 150 dan 200 *node* yaitu 15.9737ms, 42.5536ms, 77.2090ms dan 112.057ms. Ketika kecepatan *node* bertambah menjadi 40-80km/jam, nilai dari *overhead ratio* mengalami pertambahan nilai untuk *node* 50, 100, 150 dan 200 *node* yaitu 61.9000ms, 68.8000ms, 116.4074ms dan 168.7500ms. Untuk kecepatan 80-160km/jam, nilai *overhead ratio* untuk *node* 50, 100, 150 dan 200 *node* yaitu 84.0278ms, 74.0278ms, 144.9189ms dan 158.1667ms. Nilai *overhead ratio* terus bertambah naik ketika jumlah *node* yang digunakan semakin banyak. Paket *overhead ratio* merupakan parameter yang digunakan untuk

mengetahui berapa banyak *relay node* yang terkirim dari sumber ke tujuan. Pada *protocol MaxProp* nilai *overhead ratio* cenderung naik dengan ukuran *buffer* yang besar. Seperti pada pengujian ini menggunakan *buffer* sebesar 1GB.

Parameter *overhead ratio* memiliki nilai optimal ketika jumlah *node* yang digunakan semakin sedikit. Hal ini disebabkan semakin banyak nya jumlah *node* yang digunakan maka semakin banyak juga tumpukan pesan pada *buffer,* yang mengakibatkan tidak semua *node* mendapatkan pesan yang ditransmisikan dari *node* sumber.

6.3.3 Pembahasan Hasil Pengujian *Delivery Probability*

Pada tahap ini dilakukan perbandingan untuk setiap pengujian *delivery probability* dengan jumlah *node* 50, 100, 150 dan 200 *node* dengan kecepatan *node* yang berbeda-beda. Gambar 6.3 menunjukan grafik dari hasil pengujian *delivery probability* pada *protocol MaxProp,* yaitu:



Gambar 6. 2 Hasil Pengujian *Delivery Probability* Untuk *Protocol MaxProp*

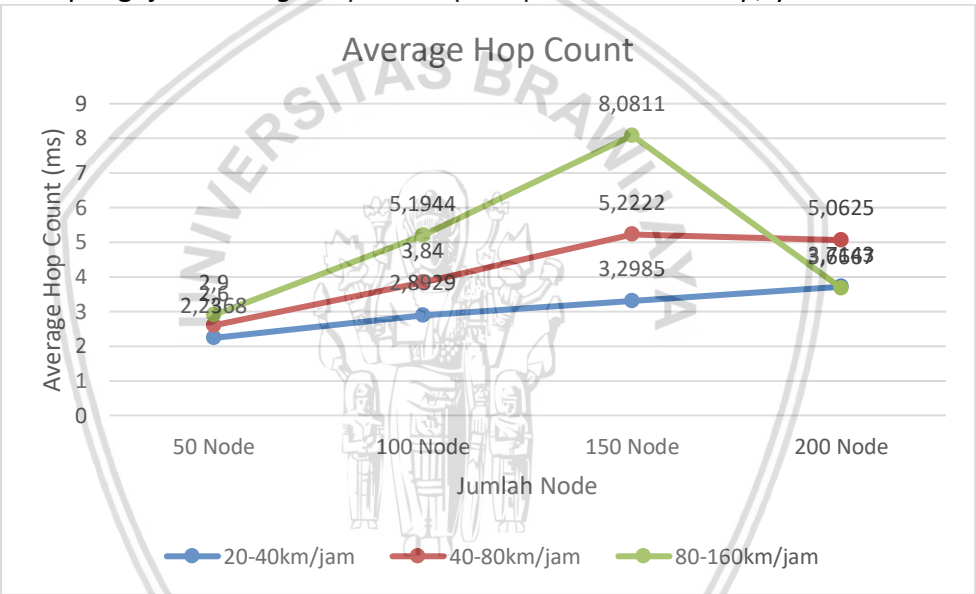
Parameter *delivery probability* adalah parameter yang menggambarkan tingkat keberhasilan pesan yang dikirimkan. Pada hasil pengujian *delivery probability,* hasil yang diperoleh untuk jumlah *node* 50, 100, 150 dan 200 *node* dan dengan kecepatan *node* 20-40km/jam, 40-80km/jam dan 80-160km/jam. Nilai *delivery probability* yang diperoleh ketika jumlah *node* yang digunakan yaitu 50, 100, 150 dan 200 *node* dengan kecepatan *node* 20-40km/jam mengalami kenaikan ketika jumlah *node* yang digunakan semakin besar yaitu 0.3115%, 0.4628%, 0.5537% dan 0.5738%.

Ketika jumlah *node* yang digunakan semakin banyak, maka nilai untuk *delivery probability* mengalami kenaikan. Hal tersebut dikarenakan semakin banyak jumlah *node* yang digunakan maka suatu pesan akan cepat sampai dan semakin banyak pesan yang akan sampai pada tujuan, karena *protocol Maxprop* dapat memperkirakan kapan suatu *node* dapat bertemu

dengan dengan menghitung *path cost calculation* dan menentukan prioritas pengiriman pesan. Dari gambar diatas diperoleh bahwa semakin banyak jumlah *node* yang digunakan, maka nilai *delivery probability* yang diperoleh semakin naik. Ketika jumlah *node* yang digunakan semakin banyak, peluang pesan sampai pada tujuan akan semakin naik. Sedangkan untuk kecepatan 40-80km/jam dengan jumlah *node* 50, 100, 150 dan 200 *node* yaitu 0.0820%, 0.2066%, 0.2231% dan 0.2025% dan untuk kecepatan *node* 80-160km/jam yaitu 0.0820%, 0.2975%, 0.3058% dan 0.1714% dengan jumlah *node* 50, 100, 150 dan 200 *node.*

6.3.4 Pembahasan Hasil Pengujian *Average Hop Count*

Pada tahap ini dilakukan perbandingan untuk setiap pengujian *average hop count* dengan jumlah *node* 50, 100, 150 dan 200 *node* dengan kecepatan *node* yang berbeda-beda. Gambar 6.4 menunjukan grafik dari hasil pengujian *average hop count* pada *protocol MaxProp,* yaitu:



Gambar 6. 3 Hasil Pengujian *Average Hop Count* Untuk *Protocol MaxProp*

Hasil pengujian diatas diperoleh dari pengujian *protocol MaxProp,* dengan jumlah *node* 50, 100, 150 dan 200 *node* dengan kecepatan *node* 20-40km/jam, 40-80km/jam dan 80-160km/jam. Dari hasil gambar diatas diperoleh nilai *average hop count* untuk jumlah *node* 50 dengan kecepatan *node* 20-40km/jam yaitu 2.2368ms. Ketika jumlah *node* yang digunakan semakin besan menjadi 100, 150 dan 200 *node* dengan kecepatan 20-40km/jam memiliki nilai *average hop count* yaitu, 2.8929ms, 3.2985ms dan 3.7143ms. Hal ini disebabkan karena dengan semakin besar ukuran *buffer* maka pesan yang ditampung akan semakin banyak. Sedangkan dengan jumlah *node* yang sedikit, memiliki kesempatan pengiriman sedikit dan pesan yang dikirimkan dilihat dari prioritas nilai *hopcount* terkecil. Sedangkan, ketika jumlah *node* yang digunakan semakin banyak, maka nilai *average hop count* cenderung mengalami penurunan.

Untuk kecepatan *node* 40-80km/jam dan dengan jumlah *node* 50 memiliki nilai *average hop count* yaitu 2.6000ms. Ketika jumlah *node* yang digunakan semakin banyak, nilai *average hop count* yang diperoleh cenderung mengalami kenaikan seperti ketika jumlah *node* yang digunakan 100, 150 dan 200 *node* memiliki nilai *average hop count* yaitu 3.8400ms, 5.2222ms dan 5.0625ms. Hal ini disebabkan oleh probabilitas suatu *node* untuk bertemu dengan *node* yang lainnya kecil. Ketika sebuah *node* memiliki kesempatan untuk mengirimkan pesan, maka pesan akan dikirimkan sesuai dengan prioritasnya. Semakin kecil bertemunya suatu *node* dengan *node* yang lain, maka saat *node* memiliki kesempatan untuk mengirimkan pesan akan langsung dikirimkan karena sedikitnya pesan dalam antrian dan nilai *hop count* setiap pesan kecil. Dan untuk kecepatan 80-160km/jam diperoleh nilai *average hop count* untuk jumlah *node* 50 memiliki nilai *average hop count* yaitu 2.9000ms. Dan ketika jumlah *node* yang digunakan ditambah menjadi 100, 150 dan 200 *node* nilai *average hop count* yang diperoleh mengalami kenaikan yaitu 5.1944ms, 8.0811ms dan 3.6667ms. Hal ini disebabkan oleh probabilitas suatu *node* untuk bertemu dengan *node* yang lainnya kecil. Ketika sebuah *node* memiliki kesempatan untuk mengirimkan pesan, maka pesan akan dikirimkan sesuai dengan prioritasnya. Semakin kecil bertemunya suatu *node* dengan *node* yang lain, maka saat *node* memiliki kesempatan untuk mengirimkan pesan akan langsung dikirimkan karena sedikitnya pesan dalam antrian dan nilai *hop count* setiap pesan kecil.

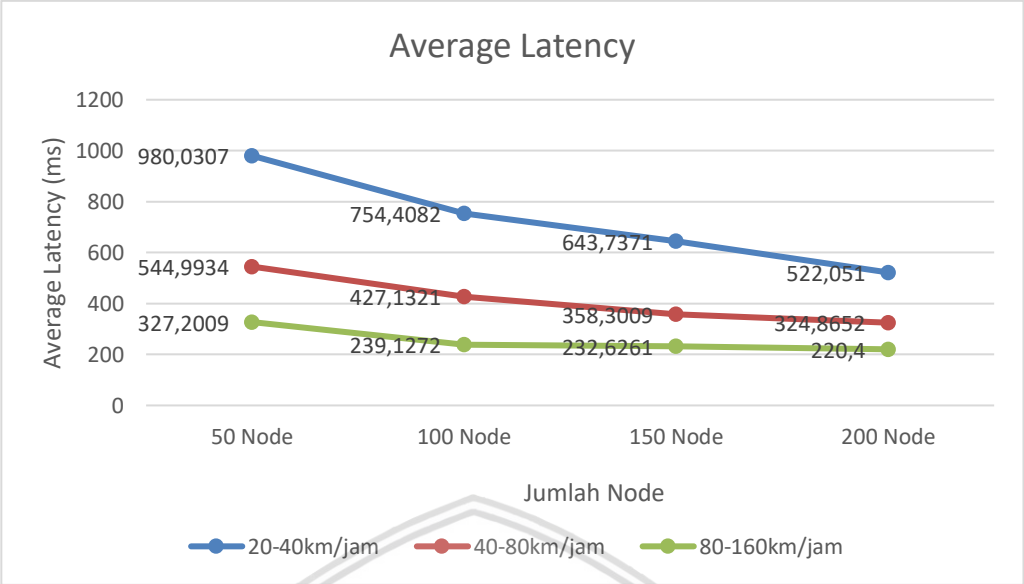
Pada *Maxprop* pengiriman pesan dipengaruhi oleh nilai *hopcount* dan pergerakan *node*. Nilai *average hop count* kecil apabila *node* sumber dengan *node* tujuan sering bertemu secara langsung. Akan tetapi nilai *average hop count* tetap cenderung turun ketika nilai *buffer* dinaikkan.

6.4 Pembahasan Hasil Pengujian *Algoritme Hierarchical Token Bucket*

Pada tahap ini, pembahasan dilakukan setelah pengujian *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy*. Hasil pengujian yang didapat berupa nilai *average latency, overhead ratio, delivery probability dan average hop count* yang ditampilkan dalam bentuk grafik agar mempermudah peneliti untuk melakukan proses analisis.

6.4.1 Pembahasan Hasil Pengujian *Average Latency*

Pada tahap ini dilakukan perbandingan untuk setiap pengujian *average latency* dengan jumlah *node* 50, 100, 150 dan 200 *node* dan kecepatan *node* yang berbeda-beda. Gambar 6.5 menunjukan grafik hasil dari pengujian *Algoritme Hierarchical Token Bucket,* yaitu:



Gambar 6. 4 Hasil Pengujian Average Latency Untuk *Algoritme Hierarchical Token Bucket* Untuk Seleksi *Node Routing Multi copy*

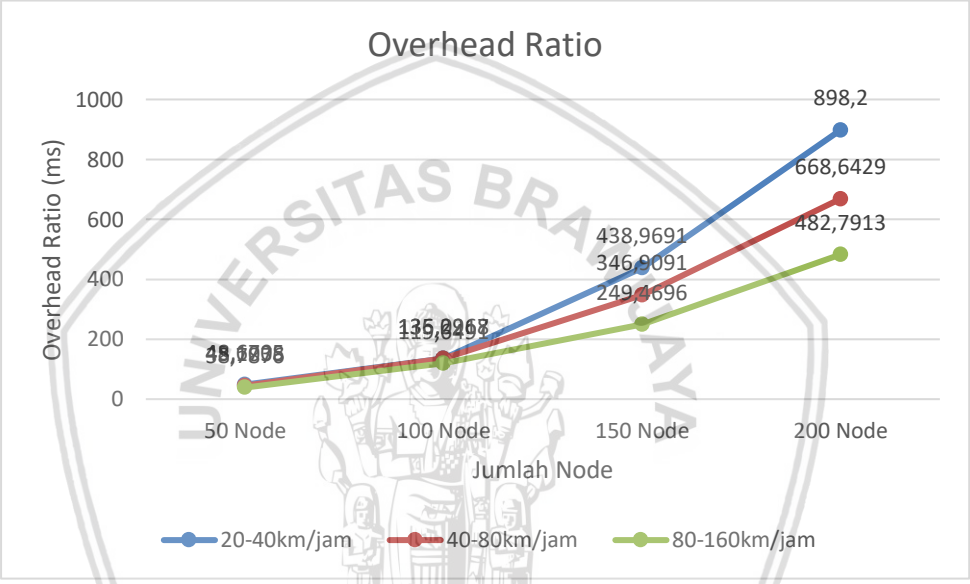
Nilai *average latency* mengalami penurunan ketika jumlah *node* yang digunakan semakin besar yaitu ketika jumlah *node* yang digunakan 50 nilai *average latency* yang diperoleh yaitu 980.0307ms. Namun mengalami penurunan nilai *average latency* ketika jumlah *node* yang digunakan 100, 150 dan 200 *node* yaitu 754.4082ms, 643.7371ms dan 522.0510ms. Nilai *average latency* dipengaruhi oleh ukuran *buffer* yang digunakan yaitu semakin besar ukuran *buffer* yang digunakan, maka nilai untuk *average latency* cenderung mengalami penurunan. Hal ini disebabkan oleh penentuan prioritas pesan yang dikirim pada *protocol MaxProp* yang bergantung pada nilai *threshold* pada *buffer*. Ketika jumlah *node* 50 dengan kecepatan *node* 40-80km/jam memiliki nilai *average latency* yaitu 544.9934ms. sedangkan ketika jumlah *node* ditambah menjadi 100, 150 dan 200 *node* nilai *average latency* yang diperoleh untuk kecepatan *node* 40-80km/jam untuk jumlah *node* 100, 150 dan 200 *node* mengalami penurunan yaitu 427.1321ms, 358.3009ms dan 324.8652ms. Hal ini disebabkan karena dengan semakin besar ukuran *buffer* maka pesan yang ditampung akan semakin banyak. Sehingga daftar antrian pada *buffer* menjadi semakin banyak. Sedangkan dengan jumlah *node* yang sedikit, memiliki kesempatan pengiriman sedikit dan dengan melihat nilai *hopcount* untuk pengiriman pesan.

Penurunan nilai pada *average latency* dengan menerapkan *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy* disebabkan oleh penggunaan *buffer* yang digunakan. Pada *Algoritme Hierarchical Token Bucket* menerapkan konsep TBF(*Token Bucket Filter)* yaitu dimana ukuran *bucket* atau pesan yang dikirim menentukan seberapa banyak *token* yang dapat disimpan. Semakin besar *buffer* yang digunakan, maka pesan yang dapat ditampung semakin banyak. Hal ini menyebabkan nilai *average latency* semakin menurun karena rata-rata waktu yang diperoleh untuk

mengirim pesan dari *node* sumber ke *node* tujuan semakin kecil. Banyaknya jumlah *node* mempengaruhi seberapa sering *node* sumber bertemu dengan *node* tujuan. Ketika *node* sumber sering bertemu dengan *node* tujuan maka nilai dari *average latency* akan naik namun jika sebaliknya maka nilai dari *average latency* akan mengalami penurunan.

6.4.2 Pembahasan Hasil Pengujian *Overhead Ratio*

Pada tahap ini dilakukan perbandingan untuk setiap pengujian *overhead ratio* dengan jumlah *node* 50, 100, 150 dan 200 *node* dengan kecepatan *node* yang berbeda-beda. Gambar 6.6 menunjukan grafik dari hasil pengujian *overhead ratio* pada *Algoritme Hierarchical Token Bucket,* yaitu:

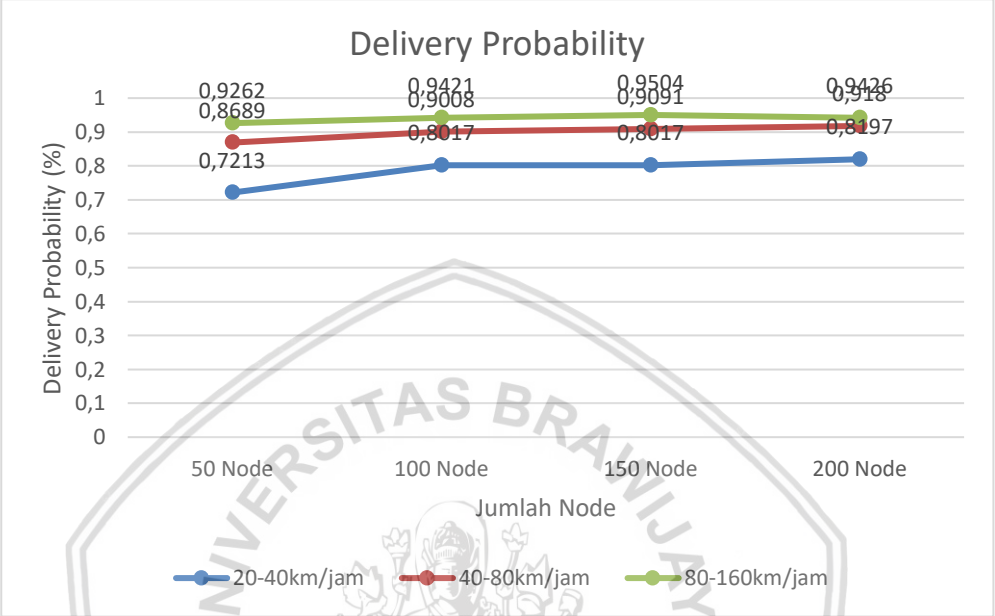


Gambar 6. 6 Hasil Pengujian *Overhead Ratio* Untuk *Algoritme Hierarchical Token Bucket*

Hasil pengujian diatas diperoleh dari pengujian *Algoritme Hierarchical Token Bucket,* dengan jumlah *node* 50, 100, 150 dan 200 *node* dengan kecepatan *node* 20-40km/jam, 40-80km/jam dan 80-160km/jam. Nilai *overhead ratio* untuk jumlah *node* 50 dengan kecepatan *node* 20-40km/jam yaitu 48.6705ms, sedangkan ketika jumlah *node* ditambah menjadi 100, 150 dan 200 *node* mengalami kenaikan yaitu 136.2268ms, 438.9691ms dan 898.2000s. Nilai *overhead ratio* terus bertambah naik ketika jumlah *node* yang digunakan semakin banyak. *Packet overhead ratio* merupakan parameter untuk mengetahui berapa banyak *relay node* yang terkirim dari sumber ke tujuan. Pada *protocol MaxProp* nilai *overhead ratio* cenderung naik dengan ukuran *buffer* yang besar. Seperti pada pengujian ini menggunakan *buffer* sebesar 1GB. Semakin banyak jumlah *node* yang digunakan, maka nilai *overhead ratio* cenderung tinggi, hal ini disebabkan oleh banyak nya daftar antrian pesan yang akan dikirimkan dari *node* sumber ke *node* tujuan.

6.4.3 Pembahasan Hasil Pengujian *Delivery Probability*

Pada tahap ini dilakukan perbandingan untuk setiap pengujian *delivery probability* dengan jumlah *node* 50, 100, 150 dan 200 *node* dengan kecepatan *node* yang berbeda-beda. Gambar 6.7 menunjukan hasil dari pengujian *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy,* yaitu:



Gambar 6. 5 Hasil Pengujian *Delivery Probability* Untuk *Algoritme Hierarchical Token Bucket* Untuk Seleksi *Node Routing Multi copy*

Hasil pengujian diatas diperoleh dari pengujian *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy,* dengan jumlah *node* 50, 100, 150 dan 200 *node* dengan kecepatan *node* 20-40km/jam, 40-80km/jam dan 80-160km/jam. Nilai *delivery probability* yang diperoleh ketika jumlah *node* yang digunakan yaitu 50, 100, 150 dan 200 *node* dengan kecepatan *node* 20-40km/jam mengalami kenaikan ketika jumlah *node* yang digunakan semakin besar yaitu 0.7213%, 0.8017%, 0.5537% dan 0.8197%.

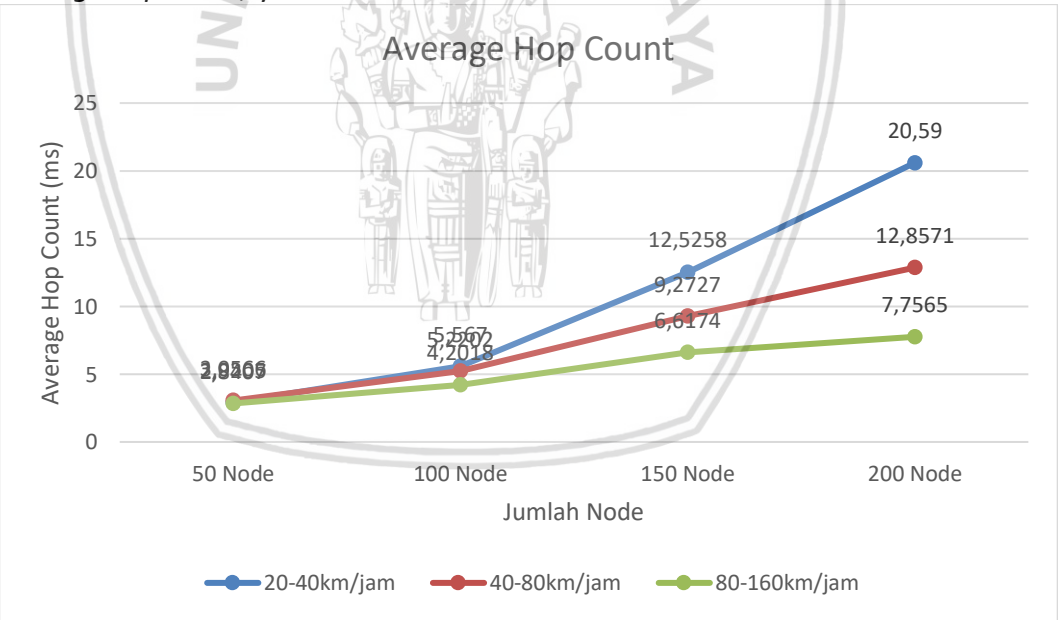
Ketika jumlah *node* yang digunakan semakin banyak, maka nilai untuk *delivery probability* mengalami kenaikan. Hal tersebut dikarenakan semakin banyak jumlah *node* yang digunakan maka suatu pesan akan cepat sampai dan semakin banyak pesan yang akan sampai pada tujuan, karena *protocol Maxprop* dapat memperkirakan kapan suatu *node* dapat bertemu dengan dengan menghitung *path cost calculation* dan menentukan prioritas pengiriman pesan. Dari gambar diatas diperoleh bahwa semakin banyak jumlah *node* yang digunakan, maka nilai *delivery probability* yang diperoleh semakin naik. Hal ini disebabkan karena, pergerakan *node* yang acak yang tidak dapat diprediksi berdasarkan jumlah *node* yang digunakan. Ketika kecepatan *node* semakin besar, peluang pesan sampai pada tujuan akan semakin naik. Sedangkan untuk kecepatan 40-80km/jam dengan jumlah *node* 50, 100, 150 dan 200 *node* yaitu 0.8689%, 0.9008%, 0.9091%

dan 0.9180% dan untuk kecepatan *node* 80-160km/jam yaitu 0.9262%, 0.9421%, 0.9504% dan 0.9426%dengan jumlah *node* 50, 100, 150 dan 200 *node*.

Kenaikan nilai *delivery probability* dipengaruhi oleh penerapan *Algoritme Hierarchical Token Bucket* pada proses pengujian. Hal ini disebabkan karena pada *Algoritme Hierarchical Token Bucket* menggunakan parameter *ceil* dimana parameter ini memungkinkan setiap *node* untuk mendapatkan *bandwidth* sesuai dengan *bandwidth* yang sudah ditentukan. Ketika jumlah *node* yang digunakan semakin banyak, maka nilai *delivery probability* mengalami kenaikan karena *Algoritme Hierarchical Token Bucket* menerapkan penjadwalan pengiriman data berdasarkan parameter *ceil* dan *rate* yang digunakan. Sehingga semua *node* akan melakukan pengiriman pesan sampai waktu yang ditentukan habis.

6.4.4 Pembahasan Hasil Pengujian *Average Hop Count*

Pada tahap ini dilakukan perbandingan untuk setiap pengujian *average hop count* dengan jumlah *node* 50, 100, 150 dan 200 *node* dengan kecepatan *node* 20-40km/jam, 40-80km/jam dan 80-160km/jam. Gambar 6.8 menunjukan grafik dari hasil pengujian pada *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy* berdasarkan parameter *average hop count,* yaitu:



Gambar 6. 6 Hasil Pengujian *Average Hop Count* Untuk *Algoritme Hierarchical Token Bucket* Untuk Seleksi *Node Routing Multi copy*

Hasil pengujian tabel 6.8 diperoleh dari pengujian *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy,* dengan jumlah *node* 50, 100, 150 dan 200 *node* dengan kecepatan *node* 20-40km/jam, 40-80km/jam dan 80-160km/jam. Dari hasil gambar diatas diperoleh nilai *average hop count* untuk jumlah *node* 50 dengan kecepatan *node* 20-40km/jam yaitu 2.9205ms. Ketika jumlah *node* yang digunakan sebesar 100, 150 dan 200 *node* dengan kecepatan 20-40km/jam yaitu,

5.5670ms, 12.5258ms dan 20.5900ms. Hal ini disebabkan oleh probabilitas suatu *node* untuk bertemu dengan *node* yang lainnya kecil. Ketika sebuah *node* memiliki kesempatan untuk mengirimkan pesan, maka pesan akan dikirimkan sesuai dengan prioritasnya. Semakin kecil bertemunya suatu *node* dengan *node* yang lain, maka saat *node* memiliki kesempatan untuk mengirimkan pesan akan langsung dikirimkan karena sedikitnya pesan dalam antrian dan nilai *hop count* setiap pesan kecil.

Untuk kecepatan *node* 40-80km/jam dan dengan jumlah *node* 50 memiliki nilai *average hop count* yaitu 3.0566ms. Dan ketika jumlah *node* yang digunakan ditamabah, nilai *average hop count* yang diperoleh cenderung mengalami pertambahan seperti ketika jumlah *node* yang digunakan 100, 150 dan 200 *node* memiliki nilai *average hop count* yaitu 5.2202ms, 9.2727ms dan 12.8571ms. Hal ini disebabkan oleh probabilitas suatu *node* untuk bertemu dengan *node* yang lainnya kecil. Ketika sebuah *node* memiliki kesempatan untuk mengirimkan pesan, maka pesan akan dikirimkan sesuai dengan prioritasnya. Semakin kecil bertemunya suatu *node* dengan *node* yang lain, maka saat *node* memiliki kesempatan untuk mengirimkan pesan akan langsung dikirimkan karena sedikitnya pesan dalam antrian dan nilai *hop count* setiap pesan kecil.

Untuk jumlah *node* 50 dengan kecepatan *node* 80-160km/jam diperoleh nilai *average hop count* yaitu 2.8407ms. Ketika jumlah *node* yang digunakan ditambah menjadi 100, 150 dan 200 *node* nilai *average hop count* yang diperoleh mengalami kenaikan yaitu 4.2018ms, 6.6174ms dan 7.7565ms. Hal ini disebabkan oleh probabilitas suatu *node* untuk bertemu dengan *node* yang lainnya kecil. Ketika sebuah *node* memiliki kesempatan untuk mengirimkan pesan, maka pesan akan dikirimkan sesuai dengan prioritasnya. Semakin kecil bertemunya suatu *node* dengan *node* yang lain, maka saat *node* memiliki kesempatan untuk mengirimkan pesan akan langsung dikirimkan karena sedikitnya pesan dalam antrian dan nilai *hop count* setiap pesan kecil.

Kenaikan nilai *average hop count* dipengaruhi oleh penerapan *Algoritme Hierarchical Token Bucket* pada proses pengujian. Hal ini disebabkan karena pada *Algoritme Hierarchical Token Bucket* menggunakan parameter *ceil* dimana parameter ini memungkinkan setiap *node* untuk mendapatkan *bandwidth* sesuai dengan *bandwidth* yang sudah ditentukan. Ketika jumlah *node* yang digunakan semakin banyak, maka nilai *average hop count* mengalami kenaikan karena *Algoritme Hierarchical Token Bucket* menerapkan penjadwalan pengiriman data berdasarkan parameter *ceil* dan *rate* yang digunakan. Sehingga semua *node* akan melakukan pengiriman pesan sampai waktu yang ditentukan habis. Parameter *rate* digunakan untuk menentukan *bandwidth* maksimum yang bisa dipakai oleh setiap *class*, jika *bandwidth* melebihi nilai "*rate*" maka pesan akan *drop*.

6.5 Total *Bandwidth* Yang Diperoleh Untuk Keseluruhan *Node* Pada Pengujian *Algoritme Hierarchical Token Bucket*

Pada tahap ini hasil diperoleh dari pengujian *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy* dengan jumlah *node* 50, 100, 150 dan 200 *node.* Banyaknya jumlah *node* akan menentukan terjadinya proses *duplicate* atau tidak selama proses peminjaman *bandwidth*. Berikut adalah hasil total *bandwidth* yang diperoleh untuk jumlah *node* 50, 100, 150 dan 200 *node,* yaitu:

6.5.1 50 Node

Total *bandwidth* yang diperoleh berdasarkan hasil pengujian menggunakan *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy.* Tabel 6.9 menampilkan hasil dari peminjaman *bandwidth* menggunakan *Algoritme Hierarchical Token Bucket* dengan jumlah 50 *node,* yaitu:

Tabel 6. 9 Total *Bandwidth* 50 *Node*

Jumlah *bandwidth* yang diminta	Jumlah *node* yang digunakan 50 *node*
1 Mbps	450 Mbps berhenti pada index *node* ke 2
2 Mbps	405 Mbps berhenti pada index *node* ke 2
3 Mbps	360 Mbps berhenti pada index *node* ke 2
4 Mbps	315 Mbps berhenti pada index *node* ke 2
5 Mbps	270 Mbps berhenti pada index *node* ke 2
6 Mbps	225 Mbps berhenti pada index *node* ke 2

Pada penggunaan *node* sebanyak 50, tidak dilakukan proses *duplicate.* Karena jumlah *bandwidth* yang diperoleh untuk keseluruhan *node* tidak melebihi dari 1GB. Ketika proses *duplicate* tidak dilakukan, maka setiap *node* memperoleh jumlah *bandwidth* sesuai yang diminta pada proses peminjaman.

Ketika jumlah *bandwidth* yang diperoleh kurang dari 1GB, hal ini sesuai dengan tujuan manajemen *bandwidth* yaitu membagi rata *bandwidth* yang disediakan untuk sejumlah *node* yang digunakan. Manajemen *bandwidth* dilakukan menggunakan *Algoritme Hierarchical Token Bucket,* dimana *Algoritme* ini memastikan setiap *node* mendapatkan *bandwidth* yang sama untuk antar *node* nya.

6.5.2. 100 Node

Total *bandwidth* yang diperoleh berdasarkan hasil pengujian menggunakan *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy.* Tabel 6.10 menampilkan hasil dari peminjaman *bandwidth* dengan jumlah 100 *node,* yaitu:

Tabel 6. 10 Total *Bandwidth* 100 *Node*

Jumlah *bandwidth* yang diminta	Jumlah *node* yang digunakan 100 *node*
1 Mbps	950 Mbps berhenti pada index *node* ke 2
2 Mbps	855 Mbps berhenti pada index *node* ke 2
3 Mbps	760 Mbps berhenti pada index *node* ke 2
4 Mbps	665 Mbps berhenti pada index *node* ke 2
5 Mbps	270 Mbps berhenti pada index *node* ke 2
6 Mbps	475 Mbps berhenti pada index *node* ke 2

Tabel diatas merupakan hasil yang diperoleh dari pengujian *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy* dengan menggunakan 100 *node.* Seperti halnya dengan 50 *node,* proses *duplicate* tidak dilakukan ketika jumlah *node* yang digunakan 100 *node,* karena total *bandwidth* yang diperoleh untuk keseluruhan *node* tidak melebihi dari ukuran *buffer* yaitu 1GB.

Hasil yang diperoleh dengan jumlah *node* sebanyak 100 *node* berfariasi sesuai dengan seberapa banyak *bandwidth* yang diminta. Semakin banyak jumlah *bandwidth* yang diminta, maka total *bandwidth* yang diperoleh semakin sedikit karena sisa *bandwidth* dari proses peminjaman yang akan didistribusikan pada *node* yang lain semakin kecil. Dari proses tersebut hasil *bandwidth* yang diperoleh akan diakumulasikan untuk keseluruhan *node* untuk mendapatkan total *bandwidth* dalam proses pengujian.

6.5.3. 150 Node

Total *bandwidth* yang diperoleh berdasarkan hasil pengujian yang dilakukan menggunakan *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy.* Tabel 6.11 menampilkan hasil dari peminjaman *bandwidth* dengan jumlah 150 *node,* yaitu:

Tabel 6. 11 Total *Bandwidth* 150 *Node*

Jumlah *bandwidth* yang diminta	Jumlah *node* yang digunakan 150 *node*
1 Mbps	1030 Mbps berhenti pada index *node* ke 86
2 Mbps	1026 Mbps berhenti pada index *node* ke 64
3 Mbps	1032 Mbps berhenti pada index *node* ke 34
4 Mbps	1015 Mbps berhenti pada index *node* ke 2
5 Mbps	870 Mbps berhenti pada index *node* ke 2
6 Mbps	725 Mbps berhenti pada index *node* ke 2

Tabel diatas merupakan hasil yang diperoleh dari pengujian *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy* dengan 150 *node.* Dari hasil pengujian diatas proses peminjaman *bandwidth* dengan jumlah 150 *node* dilakukan proses *duplicate* ketika jumlah *bandwidth* yang diminta lebih kecil dari batas maksimal *bandwidth*. Ketika jumlah *bandwidth* yang diminta sebanyak 1Mbps jumlah total *bandwidth* yang diperoleh sebanya 1030 Mbps atau melebihi dari ukuran *buffer* yang disediakan dan proses peminjaman *bandwidth* berhenti ketika berada pada *index node* ke 86 dan dilakukannya proses *duplicate*. Proses *duplicate* menguntungkan untuk hasil yang diperoleh ketika pengujian dengan jumlah *node* yang digunakan 150. Hal ini bisa dilihat pada tabel hasil pengujian 6.5, 6.6, 6.7 dan 6.8 yang mengalami kenaikan nilai ketika jumlah *node* yang digunakan semakin banyak. Hal ini terjadi karena sebagian kecil *node* yang tidak melakukan peminjaman *bandwidth* yang menyebabkan kenaikan nilai untuk beberapa parameter pengujian seperti *overhead ratio, delivery probability* dan *average hop count.*

Ketika proses *duplicate* dilakukan, maka proses peminjaman *bandwidth* yang dilakukan oleh *node* akan berhenti pada *index* tertentu. Hal ini akan berakibat pada beberapa *node* yang tidak menerima sejumlah *bandwidth* yang disediakan. *Node* yang bisa melakukan peminjaman *bandwidth* secara tidak langsung akan memperoleh pesan yang lebih banyak dibandingkan dengan *node* yang tidak bisa melakukan peminjaman *bandwidth*.

Proses *duplicate* tidak dilakukan ketika jumlah *bandwidth* yang diminta sebanyak 4, 5 dan 6 Mbps dengan memperoleh *bandwidth* sebanyak 1015 Mbps, 870 Mbps dan 725 Mbps dan proses peminjaman berakhir pada *index node* ke 2. Semakin banyak jumlah *bandwidth* yang diminta maka, total *bandwidth* yang diperoleh semakin sedikit karena sisa *bandwidth* dari proses peminjaman yang didistribusikan pada *node* yang lain semakin kecil yang berakibat pada total *bandwidth* yang diperoleh untuk keseluruhan *node* yang digunakan.

6.5.4. 200 Node

Total *bandwidth* diperoleh berdasarkan hasil pengujian menggunakan *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy.* Tabel 6.12 menampilkan hasil dari peminjaman *bandwidth* dengan jumlah 200 *node,* yaitu:

Tabel 6. 12 Total *Bandwidth* 200 *Node*

Jumlah *bandwidth* yang diminta	Jumlah *node* yang digunakan 200 *node*
1 Mbps	1030 Mbps berhenti pada index *node* ke 186
2 Mbps	1026 Mbps berhenti pada index *node* ke 124
3 Mbps	1032 Mbps berhenti pada index *node* ke 134
4 Mbps	1029 Mbps berhenti pada index *node* ke 98

5 Mbps	1026 Mbps berhenti pada index *node* ke 50
6 Mbps	975 Mbps berhenti pada index *node* ke 2

Dari hasil pengujian tabel 6.12 menunjukan hasil peminjaman *bandwidth* dengan jumlah *node* 200 *node* dan dilakukan proses *duplicate* ketika jumlah *bandwidth* yang diminta lebih kecil dari batas maksimal *bandwidth*. Ketika jumlah *bandwidth* yang diminta sebanyak 1-5 Mbps jumlah total *bandwidth* yang diperoleh sebanyak 1030 Mbps dan proses peminjaman *bandwidth* berhenti ketika berada pada *index node* ke 186, 1026 Mbps proses peminjaman *bandwidth* berhenti ketika berada pada *index node* ke 124, 1032 Mbps, proses peminjaman *bandwidth* berhenti ketika berada pada *index node* ke 134, 1029 Mbps proses peminjaman *bandwidth* berhenti ketika berada pada *index node* ke 98 dan 1026 Mbps proses peminjaman *bandwidth* berhenti ketika berada pada *index node* ke 50 atau melebihi dari ukuran *buffer* yang disediakan yaitu 1GB.

Proses *duplicate* tidak dilakukan ketika jumlah *bandwidth* yang diminta sebanyak 6 Mbps dengan memperoleh *bandwidth* sebanyak 975 Mbps dan proses peminjaman berakhir pada *index node* ke 2. Semakin banyak jumlah *bandwidth* yang diminta maka, total *bandwidth* yang diperoleh semakin sedikit karena sisa *bandwidth* dari proses peminjaman yang didistribusikan pada *node* yang lain semakin kecil yang berakibat pada total *bandwidth* yang diperoleh untuk keseluruhan *node* yang digunakan. Semakin banyak jumlah *node* yang digunakan, maka ukuran *buffer* yang digunakan semakin besar.

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan hasil dari perancangan, implementasi, pengujian, dan analisis yang telah dilakukan pada *protocol MaxProp* dan *Algoritme Hierarchical Token Bucket* diperoleh kesimpulan sebagai berikut:

1. Implementasi *Algoritme Hierarchical Token Bucket* pada penelitian ini dilakukan dengan menggunakan simulasi *ONE Simulator*. Seleksi *node* dilakukan terlebih dahulu menggunakan mekanisme *two hop routing* yaitu pesan hanya akan dikirimkan pada *node* yang menjadi tujuan dari *node* sumber. Pada penelitian ini mekanisme *two hop routing* dibagi berdasarkan *index node* ganjil dan *node* genap. Implementasi dilakukan dengan menggunakan bahasa pemrograman *JAVA* pada *Eclipse* dengan menggunakan jumlah *node* 50, 100, 150 dan 200 *node,* dengan kecepatan *node* 20-40km/jam, 40-80km/jam dan 80-160km/jam dan dengan ukuran pesan 1MB.
2. Implementasi *protocol MaxProp* dilakukan dengan melakukan konfigurasi *file default setting* pada *ONE Simulator* dengan jumlah *node* 50, 100, 150 dan 200 *node,* dengan kecepatan *node* 20-40km/jam, 40-80km/jam dan 80-160km/jam dan ukuran pesan 1MB.
3. Pada penelitian ditemukan proses *duplicate* yaitu berhentinya proses peminjaman *bandwidth,* ketika total *bandwidth* yang diperoleh melebihi dari 1GB pada skenario jumlah *node* 150 dan 200 *node.*
4. Hasil pengujian pada parameter *average latency* diperoleh bahwa hasil pengujian dari *protocol MaxProp* tanpa menggunakan *Algoritme Hierarchical Token Bucket* memiliki nilai *average latency* paling baik dengan jumlah *node* 50, 100, 150 dan 200 yaitu 1134.8947ms, 1072.5196ms, 1058.6030ms dan 1074.9700.
5. Hasil pengujian dari parameter *overhead ratio* diperoleh bahwa hasil dari pengujian *protocol MaxProp* memiliki nilai *overhead ratio* paling baik yaitu 15.9737ms, 42.5536ms, 61.9000ms dan 68.8000ms.
6. Hasil pengujian dari parameter *delivery probability* diperoleh bahwa hasil dari pengujian *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy* memiliki nilai paling baik. Dengan nilai 6% lebih besar dibandingkan pada pengujian *protocol MaxProp* yaitu 0.7213%, 0.8017%, 0.8017% dan 0.8197%.
7. Hasil pengujian dari parameter *average hop count* diperoleh bahwa hasil dari pengujian *Algoritme Hierarchical Token Bucket* untuk seleksi *node routing multi copy* memiliki nilai paling baik yaitu 2.9205ms, 5.5670ms, 12.5258ms dan 20.5900ms.

Dari poin-poin diatas yang dijelaskan secara umum, diperoleh kesimpulan bahwa, *Algoritme Hierarchical Token Bucket* dapat diimplementasikan dengan baik pada penelitian ini dengan skenario jumlah *node* 50, 100, 150 dan 200, kecepatan *node* 20-160km/jam dan ukuran pesan 1MB berdasarkan hasil dari pengujian parameter *delivery probability*

dan *average hop count.* Sedangkan untuk parameter *average latency* dan *overhead ratio, protocol maxprop* dapat diimplementasikan dengan baik pada penelitian ini dengan skenario jumlah *node* 50, 100, 150 dan 200, kecepatan *node* 20-160km/jam dan ukuran pesan 1MB.

7.2 Saran

Saran yang diperoleh dari hasil penelitian *Implementasi Delay Toelrant Network* Dengan Menggunakan *Algoritme Hierarchical Token Bucket* Untuk Seleksi *Node Routing Multi copy* adalah sebagai berikut:

1. Dapat dilakukan pengembangan terhadap seleksi *node* dengan menggunakan mekanisme *tree based flooding* terhadap *protocol multi copy* yang lain seperti *spray and wait* dan RAPID pada *Delay Tolerant Network.*
2. Dapat dilakukan analisis dengan menambahkan parameter *manajement buffer* supaya hasil yang diperoleh memiliki variasi nilai berdasarkan dari seberapa besar jumlah *buffer* yang digunakan.
3. Dapat dilakukan implementasi dengan variasi jumlah *node* yang lebih banyak dan parameter pengujian yang lain.

DAFTAR PUSTAKA

Abdillah, Y. A., Wibowo, T. A. & Yovita, L. V., 2015. ANALISIS PERFORMANSI ROUTER MAXPROP PADA VEHICULAR AD HOC NETWORK BERBASIS DELAY TOLERANT NETWORK. *Fakultas Teknik Elektro, Universitas Telkom.*

Alaoui, E. A. A., Agoujil, S., Hajar, M. & Qaraai, Y., 2015. The Performance of DTN Routing Protocols: A Comparative Study. *Department of Computer Science Faculty of Sciences and Technology Errachidia.*

Bindra, H. S. & Sangal, A. L., 2012. Performance Comparison of RAPID, Epidemic and Prophet Routing Protocols for Delay Tolerant Networks. *International Journal of Computer Theory and Engineering.*

Caini, C., Firrincieli, R. & Livini, M., 2010. DTN Bundle Layer over TCP : Retransmission Alghorithm in the Presence of Channel Disruptions. *Jurnal Of Communications,* Volume 5.

Hariyadi, C., 2009. Graf Dalam Topologi Jaringan. *Program Studi Teknik Informatika Institut Teknologi Bandung.*

Irianto, J. F., 2014. Pengertian dan Jenis-Jenis Topologi Jaringan. *Ilmu Teknologi Informasi.*

Jones, E. P. & Ward, P. A., 2006. Routing Strategies for Delay-Tolerant Networks. *University of Waterloo 200 University Avenue West Waterloo, Ontario, Canada.*

Kimura, T. & Premachandra, C., 2016. Optimal Relay Node Selection in Two-Hop Routing for Intermittently Connected MANETs. *Tokyo University of Science.*

Krifa, A. & Barakat, C., 2008. An Optimal Joint Scheduling and Drop Policy for Delay Tolerant Networks. *Swiss Federal Institute of Technology (ETH)*.

Liu, J., Jiang, X., Nishiryama, H. & Kato, N., 2011. A General Model for Store-carry-forward Routing Schemes with Multicasr in Delay Tolerant Network.

Muis, A., Niswar, M. & Ilham, A. A., 2018. OPTIMISASI KINERJA MANAJEMEN BUFFER PADA JARINGAN DELAY TOLERANT NETWORK (DTN) UNTUK JENIS ROUTING MULTI COPY. *Jurusan Teknik Informatika,Universitas Indonesia Timur Makassar.*

Ngoen, T. S., 2003. AVL Tree. *Universitas Bina Nusantara Fakultas Ilmu Komputer.*

Nugraha, I., 2016. Indahnya Jalur Lingkar Selatan Dari Sendangbiru ke Balekambang. *www.kompasiana.com*.

Patel, C. M. & Gondaliya, N., 2015. Enhancement of Social based Routing Protocol in Delay Tolerant Networks. *International Journal of Computer Applications.*

Raj, V. S. & Chezian, D. R. M., 2013. DELAY – Disruption Tolerant Network (DTN), its Network Characteristics and Core Applications. *International Journal of Computer Science and Mobile Computing.*

Siregar, R. R., 2009. Bandwidth Management Dengan Hierarchical Token Bucket (HTB) Di mesin Linux.. *Teknik Informatika FTI Universitas Trisakti*.

Wahyuni, T., Tola, M. & Niswar, M., t.thn. DATA TRANSMISSION PERFOMANCE ON DTN BY MEANS OF MULTI COPY ROUTING. *Jurusan Teknik Informatika,Universitas Indonesia Timur Makassar.*

Wang, H., Liu, X., Hu, X. & Liu, Q., 2010. The Mobile Scenario Influence On Delay Tolerant Network Routing. *School Of Software Beihang University.*

Wang, w. et al., 2015. Research on Routing Protocols and Simulation Analysis for Opportunistic Networks. *International Journal of Multimedia and Ubiquitous Engineering,* Volume 10.

Wijaya, A. I. & Handoko, L. B., 2013. MANAJEMEN BANDWIDTH DENGAN METODE HTB (HIERARCHICAL TOKEN BUCKET) PADA SEKOLAH MENENGAH PERTAMA NEGERI 5 SEMARANG. *JURNAL TEKNIK INFORMATIKA UDINUS.*

Yang, Z., Huang, L., Xiao, M. & Liu, W., 2012. Flow-Based Transmission Scheduling in Constrained Delay Tolerant Networks. *Department of Computer Science & Technology*.